

clockControl.c

```

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 {
27
28 // 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;

```

clockControl.c

```

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         //that's when we want to actually start auto-updating
126             currentState = rate_timer_running_st;
127             clockDisplay_performIncDec(); //we begin the auto-update
process with a single inc/dec. The rest is taken care of in the next state
128         }
129         break;
130     case rate_timer_running_st: //this is one of the two states that actually handles
the auto-updating
131         //this first part is to check to see if the display is
still being continuously touched
132         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
133             currentState = waiting_for_touch_st;
134         }

```

clockControl.c

```

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 user 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 is 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/dec 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

```

clockControl.c

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

clockDisplay.c

```
1 #include "clockDisplay.h"
2 #include "supportFiles/display.h" //needed to access the LCD screen
3 #include "stdio.h" //needed to make printf work
4 #include "supportFiles/utils.h"
5 #include "string.h"
6
7 #define CLOCK_TEXT_SIZE 6
8 #define HALF_SCREEN_HEIGHT DISPLAY_HEIGHT/2 //half the LCD display height
9 #define HALF_SCREEN_WIDTH DISPLAY_WIDTH/2 //half the LCD width
10 #define ONE_THIRD_WIDTH DISPLAY_WIDTH/3
11 #define TWO_THIRDS_WIDTH (DISPLAY_WIDTH*2)/3
12 #define RUNNING 1
13 #define CLOCK_DELAY 1000
14 #define TOUCH_DELAY 45
15
16 #define TRI_HEIGHT_OFFSET 17
17 //multiply by text size to get height of triangles relative to display center
17 #define LOWER_TRI_HEIGHT
18 HALF_SCREEN_HEIGHT+(CLOCK_TEXT_SIZE*TRI_HEIGHT_OFFSET)//y-coord of lowest point on
19 lower triangles
18 #define UPPER_TRI_HEIGHT HALF_SCREEN_HEIGHT-
19 (CLOCK_TEXT_SIZE*TRI_HEIGHT_OFFSET)//y-coord of highest point on higher triangles
19 #define TRI_BASE_OFFSET 8
20 //multiply by text size to get y-coord of triangle bases
20 #define LOWER_TRI_BASE HALF_SCREEN_HEIGHT+(CLOCK_TEXT_SIZE*TRI_BASE_OFFSET)
21 //y-coord of base of lower triangles
21 #define UPPER_TRI_BASE HALF_SCREEN_HEIGHT-(CLOCK_TEXT_SIZE*TRI_BASE_OFFSET)
22 //y-coord of base of upper triangles
22
23 #define LEFT_LEFT_OFFSET 24 //multiply
24 by text size to get x-coord of left point of left triangles
24 #define LEFT_LEFT_X HALF_SCREEN_WIDTH-(CLOCK_TEXT_SIZE*LEFT_LEFT_OFFSET) //x-coord
25 of left point of left triangles
25 #define LEFT_MIDDLE_OFFSET 18 //multiply
26 by text size to get x-coord of middle point of left triangles
26 #define LEFT_MIDDLE_X HALF_SCREEN_WIDTH-(CLOCK_TEXT_SIZE*LEFT_MIDDLE_OFFSET) //x-coord
27 of middle point of left triangles
27 #define LEFT_RIGHT_OFFSET 12 //multiply
28 by text size to get x-coord of right point of left triangles
28 #define LEFT_RIGHT_X HALF_SCREEN_WIDTH-(CLOCK_TEXT_SIZE*LEFT_RIGHT_OFFSET) //x-coord
29 of right point of left triangles
29 #define MIDDLE_LEFT_OFFSET 6 //multiply
30 by text size to get x-coord of left point of middle triangles
30 #define MIDDLE_LEFT_X HALF_SCREEN_WIDTH-(CLOCK_TEXT_SIZE*MIDDLE_LEFT_OFFSET) //x-coord
31 of left point of middle triangle
31
32 #define MIDDLE_RIGHT_OFFSET 6
33 //multiply by text size to get x-coord of right point of middle triangles
33 #define MIDDLE_RIGHT_X
34 HALF_SCREEN_WIDTH+(CLOCK_TEXT_SIZE*MIDDLE_RIGHT_OFFSET)//x-coord of right point of
35 middle triangles
34 #define RIGHT_LEFT_OFFSET 12
35 //multiply by text size to get x-coord of left point of right triangles
35 #define RIGHT_LEFT_X HALF_SCREEN_WIDTH+(CLOCK_TEXT_SIZE*RIGHT_LEFT_OFFSET)
36 //x-coord of left point of right triangles
36 #define RIGHT_MIDDLE_OFFSET 18
37 //multiply by text size to get x-coord of middle point of right triangles
37 #define RIGHT_MIDDLE_X
```

clockDisplay.c

```

    HALF_SCREEN_WIDTH+(CLOCK_TEXT_SIZE*RIGHT_MIDDLE_OFFSET)//x-coord of middle point of
    right triangles
38 #define RIGHT_RIGHT_OFFSET 24
    //multiply by text size to get x-coord of right point of right triangles
39 #define RIGHT_RIGHT_X HALF_SCREEN_WIDTH+(CLOCK_TEXT_SIZE*RIGHT_RIGHT_OFFSET)
    //x-coord of right point of right triangles
40
41 #define TEXT_CURSOR_OFFSET_Y 4 //multiply by
    text size to get y position for text cursor
42 #define TEXT_Y HALF_SCREEN_HEIGHT-(CLOCK_TEXT_SIZE*TEXT_CURSOR_OFFSET_Y)//y position
    for text cursor
43 #define CURSOR_MULTIPLIER 6 //scaling
    factor needed to get the right cursor x-coord
44 //when
    comparing oldBuffer and newBuffer character by character
45
46 #define INC_HOURS 0 //identifier assigned to top left rectangle of display
47 #define DEC_HOURS 1 //identifier assigned to bottom left rectangle of display
48 #define INC_MIN 2 //identifier assigned to top middle rectangle of display
49 #define DEC_MIN 3 //identifier assigned to bottom middle rectangle of display
50 #define INC_SEC 4 //identifier assigned to top right rectangle of display
51 #define DEC_SEC 5 //identifier assigned to bottom right rectangle of display
52 #define NO_TOUCH_ERROR -1 //used to determine if a touch was detected even though it
    was not actually touched
53
54 //all values based on 12-hour clock model
55 #define HOURS_MIN 1 //lowest value the hours can have
56 #define MIN_MIN 0 //lowest value the minutes can have
57 #define SEC_MIN 0 //lowest value the seconds can have
58 #define HOURS_MAX 12 //max val for hours
59 #define MIN_MAX 59 //max val for minutes
60 #define SEC_MAX 59 //max val for seconds
61
62 #define ERROR_STRING "TOUCH ERROR\n\r"
63
64 #define BUFFER_SIZE 9 //the size of the clock strings we will be displaying
65
66 static int8_t hours = HOURS_MIN; //initialize hours to 1
67 static int8_t min = MIN_MIN; //initialize minutes to 0
68 static int8_t sec = SEC_MIN; //initialize seconds to 0
69 static char newBuffer[9]; //this will hold the updated string after an inc/dec
    function call or second increment
70 static char oldBuffer[9]; //this holds a copy of the previous string in order
    to compare it with the new one
71
72
73
74
75 // Called only once - performs any necessary inits.
76 // This is a good place to draw the triangles and any other
77 // parts of the clock display that will never change.
78 void clockDisplay_init(){
79     display_init(); //Must init all of the software and
    underlying hardware for LCD.
80     display_fillScreen(DISPLAY_BLACK); //Blank the screen.
81     display_setTextSize(CLOCK_TEXT_SIZE); //set text size
82     display_setTextColor(DISPLAY_GREEN); //text color given by lab specs
83

```

clockDisplay.c

```

84     display_fillTriangle(LEFT_LEFT_X, UPPER_TRI_BASE, LEFT_MIDDLE_X, UPPER_TRI_HEIGHT,
    //create the triangle for incrementing hours
85         LEFT_RIGHT_X, UPPER_TRI_BASE, DISPLAY_GREEN);
86     display_fillTriangle(MIDDLE_LEFT_X, UPPER_TRI_BASE, HALF_SCREEN_WIDTH,
    UPPER_TRI_HEIGHT, //create the triangle for incrementing minutes
87         MIDDLE_RIGHT_X, UPPER_TRI_BASE, DISPLAY_GREEN);
88     display_fillTriangle(RIGHT_LEFT_X, UPPER_TRI_BASE, RIGHT_MIDDLE_X,
    UPPER_TRI_HEIGHT, //create the triangle for incrementing seconds
89         RIGHT_RIGHT_X, UPPER_TRI_BASE, DISPLAY_GREEN);
90
91     display_fillTriangle(LEFT_LEFT_X, LOWER_TRI_BASE, LEFT_MIDDLE_X, LOWER_TRI_HEIGHT,
    //create triangle for decrementing hours
92         LEFT_RIGHT_X, LOWER_TRI_BASE, DISPLAY_GREEN);
93     display_fillTriangle(MIDDLE_LEFT_X, LOWER_TRI_BASE, HALF_SCREEN_WIDTH,
    LOWER_TRI_HEIGHT, //create triangle for decrementing minutes
94         MIDDLE_RIGHT_X, LOWER_TRI_BASE, DISPLAY_GREEN);
95     display_fillTriangle(RIGHT_LEFT_X, LOWER_TRI_BASE, RIGHT_MIDDLE_X,
    LOWER_TRI_HEIGHT, //create triangle for decrementing seconds
96         RIGHT_RIGHT_X, LOWER_TRI_BASE, DISPLAY_GREEN);
97
98     sprintf(oldBuffer, "%2hd:%02hd:%02hd", hours, min, sec); //what will first appear
    on the screen when initialized
99     display_setCursor(LEFT_LEFT_X, TEXT_Y); //set cursor based on
    display midpoint
100     display_print(oldBuffer); //printing to the display
101 }
102
103 // Updates the time display with latest time, making sure to update only those digits
    that
104 // have changed since the last update.
105 // if forceUpdateAll is true, update all digits.
106 void clockDisplay_updateTimeDisplay(bool forceUpdateAll){
107     if(newBuffer != oldBuffer){ //only need to update time on display if the time to
    be displayed is different from current time
108         for(int i = 0; i < BUFFER_SIZE; i++){ //for loop to go through the two buffers
    character by character
109             if(oldBuffer[i] != newBuffer[i]){ //only enter if statement if the
    characters at ith position in buffer are different
110                 display_drawChar(LEFT_LEFT_X+(i*CLOCK_TEXT_SIZE)*CURSOR_MULTIPLIER,
    TEXT_Y, oldBuffer[i], DISPLAY_BLACK, DISPLAY_BLACK, CLOCK_TEXT_SIZE); //black out the
    older character so it can't be seen
111
112                 display_drawChar(LEFT_LEFT_X+(i*CLOCK_TEXT_SIZE)*CURSOR_MULTIPLIER,
    TEXT_Y, newBuffer[i], DISPLAY_GREEN, DISPLAY_BLACK, CLOCK_TEXT_SIZE); //draw over old
    character with green text
113             }
114         }
115         strcpy(oldBuffer, newBuffer); //built in string function to copy the updated
    buffer just put on display to the old buffer
116     }
117 }
118
119
120 //helper function to get the coordinates of the touch and determine which region of
    the display it is
121 int8_t getQuadrant(int16_t x, int16_t y){ //we need the x and y-coords to determine
    region of the display
152

```

clockDisplay.c

```
156
157 //helper function for ensuring the sec, min, and hours rollover when necessary
158 void incDecRollover() {
159     if(sec > SEC_MAX){ //are the seconds greater than 59?
160         sec = SEC_MIN; //set it back to 0
161         if(min == MIN_MAX){ //also check are the minutes at 59?
162             min = MIN_MIN; //set them back to 0
163             hours++; //in which case we also need to increment the hours
164         }
165     } else{
166         min++; //if the minutes aren't yet at 59, increment the minutes
167     }
168 }
169 else if(sec < SEC_MIN){ //are the seconds less than 0?
170     sec = SEC_MAX; //set it back to 59
171     if(min == MIN_MIN){ //also check if minutes are at 0
172         min = MIN_MAX; //if so, set them back to 59
173         hours--; //in which case we also need to decrement the hours
174     }
175 } else{
176     min--; //if the minutes aren't yet at 0, just decrement them by 1
177 }
178 }
179
180
181 if(min > MIN_MAX){ //are the minutes greater than 59?
182     min = MIN_MIN; //set them back to 0
183     if(hours == HOURS_MAX){ //also check if hours are at 12
184         hours = HOURS_MIN; //if so, set that back to 1
185     }
186 } else{
187     hours++; //if the hours aren't at 12 yet, just increment them by 1
188 }
189 }
190 else if(min < MIN_MIN){ //are minutes less than 0?
191     min = MIN_MAX; //set them back to 59
192     if(hours == HOURS_MIN){ //also check if hours are at 1
193         hours = HOURS_MAX; //if so set them back to 12
194     }
195 } else{
196     hours--; //if hours aren't at 1 yet, just decrement them by 1
197 }
198 }
199
200 //final hours check after incrementing or decrementing may have happened
201 if(hours > HOURS_MAX){ //are hours greater than 12?
202     hours = HOURS_MIN; //set them back to 1
203 }
204 else if(hours < HOURS_MIN){ //are hours less than 1?
205     hours = HOURS_MAX; //set them back to 12
206 }
207 }
208
209 // Reads the touched coordinates and performs the increment or decrement,
210 // depending upon the touched region.
211 void clockDisplay_performIncDec() {
212     int16_t x, y; //need x,y coordinates
213     uint8_t z; //need z for touch pressure as argument for function. Will not be
```


clockDisplay.c

```
used again.
214 display_getTouchedPoint(&x, &y, &z); //used to get the x,y coordinates for the
touch
215 int8_t whereTouched = getQuadrant(x,y); //whereTouched is assigned to the correct
region of the display based on
216
217 switch(whereTouched){ //switch to check the quadrant and carry out corresponding
action
218 case INC_HOURS:
219     hours++; //increments hours if the top left region was touched
220     break;
221 case DEC_HOURS:
222     hours--; //decrements hours if the bottom left region was touched
223     break;
224 case INC_MIN:
225     min++; //increments minutes if top middle region was touched
226     break;
227 case DEC_MIN:
228     min--; //decrements minutes if bottom middle region was touched
229     break;
230 case INC_SEC:
231     sec++; //increments seconds if top right region was touched
232     break;
233 case DEC_SEC:
234     sec--; //decrements seconds if bottom right region was touched
235     break;
236 case NO_TOUCH_ERROR:
237     printf(ERROR_STRING); //if there was a touch error, then print it out
238     break;
239 default:
240     break;
241 }
242
243 incDecRollover(); //after the right thing is incremented or decremented, we need
to check for rollover
244 sprintf(newBuffer, "%2hd:%02hd:%02hd", hours, min, sec); //send the updated
hours/minutes/seconds into the newBuffer character array
245 clockDisplay_updateTimeDisplay(false); //now we update the display with the
updated time character array
246 }
247
248 // Advances the time forward by 1 second and update the display.
249 void clockDisplay_advanceTimeOneSecond(){
250     sec++; //always increment the seconds first. All the checks for whether it should
actually be incremented or not have already been done up to this point
251     incDecRollover(); //we have to check for rollover again in case it changed the
minutes as well
252     sprintf(newBuffer, "%2hd:%02hd:%02hd", hours, min, sec); //send the updated
hours/minutes/seconds into the newBuffer character array
253     clockDisplay_updateTimeDisplay(false); //now we update the display with the
updated time character array
254 }
255
256 // Run a test of clock-display functions.
257 void clockDisplay_runTest(){
258     clockDisplay_init(); //must initialize the clock display, which includes the
triangles and setting the time as 1:00:00
259 }
```

clockDisplay.c

```
260     while(RUNNING){ //run as long as we want it to, constantly looking for new touch
input
261         clockDisplay_performIncDec(); //this will translate the touch into which part
of the time to be incremented or decremented
262         clockDisplay_advanceTimeOneSecond(); //this must be happening whether we are
getting touch input or not
263     }
264 }
265
```