

clockMain.c

```
2 * clockMain.c
7
8 /*****
9 *****/
10 *****/
11 #include <stdio.h>
12 #include "supportFiles/leds.h"
13 #include "supportFiles/globalTimer.h"
14 #include "supportFiles/interrupts.h"
15 #include "supportFiles/utils.h"
16 #include <stdbool.h>
17 #include <stdint.h>
18 #include "clockControl.h"
19 #include "clockDisplay.h"
20 #include "supportFiles/display.h"
21
22 #include "xparameters.h"
23
24 #define TOTAL_SECONDS 60
25 // The formula for computing the load value is based upon the formula from 4.1.1
26 // (calculating timer intervals)
27 // in the Cortex-A9 MPCore Technical Reference Manual 4-2.
28 // Assuming that the prescaler = 0, the formula for computing the load value based
29 // upon the desired period is:
30 // load-value = (period * timer-clock) - 1
31 #define TIMER_PERIOD 91.0E-3 // You can change this value to a value that you
32 // select.10.0E-3
33 #define TIMER_CLOCK_FREQUENCY (XPAR_CPU_CORTEXA9_0_CPU_CLK_FREQ_HZ / 2)
34 #define TIMER_LOAD_VALUE ((TIMER_PERIOD * TIMER_CLOCK_FREQUENCY) - 1.0)
35
36 /*
37 int main() {
38     clockDisplay_init();
39     clockControl_init();
40     while (1) {
41         clockControl_tick();
42         utils_msDelay(100);
43     }
44 }
45 */
46
47 int main()
48 {
49     // Initialize the GPIO LED driver and print out an error message if it fails
50     // (argument = true).
51     // You need to init the LEDs so that LD4 can function as a heartbeat.
52     leds_init(true);
53     // Init all interrupts (but does not enable the interrupts at the devices).
54     // Prints an error message if an internal failure occurs because the argument =
55     // true.
56     interrupts_initAll(true);
57     interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
58     u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
59     printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
60     // Allow the timer to generate interrupts.
61     interrupts_enableTimerGlobalInts();
62     // Initialization of the clock display is not time-dependent, do it outside of the
63     // state machine.
```

clockMain.c

```
58     clockDisplay_init();
59     // Keep track of your personal interrupt count. Want to make sure that you don't
    miss any interrupts.
60     int32_t personalInterruptCount = 0;
61     // Start the private ARM timer running.
62     interrupts_startArmPrivateTimer();
63     // Enable interrupts at the ARM.
64     interrupts_enableArmInts();
65     // interrupts_isrInvocationCount() returns the number of times that the timer ISR
    was invoked.
66     // This value is maintained by the timer ISR. Compare this number with your own
    local
67     // interrupt count to determine if you have missed any interrupts.
68     while (interrupts_isrInvocationCount() < (TOTAL_SECONDS *
    privateTimerTicksPerSecond)) {
69         if (interrupts_isrFlagGlobal) { // This is a global flag that is set by the
    timer interrupt handler.
70             // Count ticks.
71             personalInterruptCount++;
72             clockControl_tick();
73             interrupts_isrFlagGlobal = 0;
74         }
75     }
76     interrupts_disableArmInts();
77     printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
78     printf("internal interrupt count: %ld\n\r", personalInterruptCount);
79     return 0;
80 }
81 ///
82
83 void isr_function() {
84
85     // Leave blank for the flag method.
86     // I already set the flag for you in another routine.
87 }
88 */
89
90
91 static uint32_t isr_functionCallCount = 0;
92
93 int main()
94 {
95     // Initialize the GPIO LED driver and print out an error message if it fails
    (argument = true).
96     // You need to init the LEDs so that LD4 can function as a heartbeat.
97     leds_init(true);
98     // Init all interrupts (but does not enable the interrupts at the devices).
99     // Prints an error message if an internal failure occurs because the argument =
    true.
100    interrupts_initAll(true);
101    interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
102    printf("timer load value: %ld\n\r", (int32_t) TIMER_LOAD_VALUE);
103    u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
104    printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
105    interrupts_enableTimerGlobalInts();
106    // Initialization of the clock display is not time-dependent, do it outside of the
    state machine.
107    clockDisplay_init();
```

clockMain.c

```
108 // Start the private ARM timer running.
109 interrupts_startArmPrivateTimer();
110 // Enable interrupts at the ARM.
111 interrupts_enableArmInts();
112 // The while-loop just waits until the total number of timer ticks have occurred
    before proceeding.
113 while (interrupts_isrInvocationCount() < (TOTAL_SECONDS *
    privateTimerTicksPerSecond));
114 // All done, now disable interrupts and print out the interrupt counts.
115 interrupts_disableArmInts();
116 printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
117 printf("internal interrupt count: %ld\n\r", isr_functionCallCount);
118 return 0;
119 }
120
121 // The clockControl_tick() function is now called directly by the timer interrupt
    service routine.
122 void isr_function() {
123     clockControl_tick();
124     isr_functionCallCount++;
125     // Add the necessary code here.
126 }
127
128
```

clockDisplay.h

```
2  * clockDisplay.h
7
8 #ifndef CLOCKDISPLAY_H_
9 #define CLOCKDISPLAY_H_
10
11 #include <stdbool.h>
12
13 // Called only once - performs any necessary inits.
14 // This is a good place to draw the triangles and any other
15 // parts of the clock display that will never change.
16 void clockDisplay_init();
17
18 // Updates the time display with latest time, making sure to update only those digits
   that
19 // have changed since the last update.
20 // if forceUpdateAll is true, update all digits.
21 void clockDisplay_updateTimeDisplay(bool forceUpdateAll);
22
23 // Reads the touched coordinates and performs the increment or decrement,
24 // depending upon the touched region.
25 void clockDisplay_performIncDec();
26
27 // Advances the time forward by 1 second and update the display.
28 void clockDisplay_advanceTimeOneSecond();
29
30 // Run a test of clock-display functions.
31 void clockDisplay_runTest();
32
33
34 #endif /* CLOCKDISPLAY_H_ */
35
```

clockDisplay.c

```
2 * clockDisplay.c
7#include "clockDisplay.h" //include the header file for this .c file
8#include "supportFiles/display.h" //include the display header so you can display to
the board
9#include "supportFiles/utils.h" //this includes a file that makes the clock work
10#include "stdio.h"
11//#include <stdio.h>
12
13#define CLK_TEXT_SIZE 6 // set the text size to 6
14#define CLOCK_SEPERATOR ":" //is the colon to separate the hours min and sec
15#define DISPLAY_MIDDLE_X (DISPLAY_WIDTH / 2) //finds the middle of the board in the X
direction 160 pixels
16#define DISPLAY_MIDDLE_Y (DISPLAY_HEIGHT / 2) //finds the middle of the board in the Y
direction 120 pixels
17#define LINE_0_X (DISPLAY_WIDTH / 3)
18#define LINE_1_X (2 * (DISPLAY_WIDTH / 3))
19
20#define TEXT_0_CURSOR_X (DISPLAY_MIDDLE_X - ((135/6) * CLK_TEXT_SIZE)) //X location
for hours
21#define TEXT_1_CURSOR_X (DISPLAY_MIDDLE_X - ((30/6) * CLK_TEXT_SIZE)) //X
location for minutes
22#define TEXT_2_CURSOR_X (DISPLAY_MIDDLE_X + ((75/6) * CLK_TEXT_SIZE)) //X location
for seconds
23#define TEXT_0_COLON_X (DISPLAY_MIDDLE_X - ((70/6) * CLK_TEXT_SIZE)) //X location
for hours to minutes colon 65
24#define TEXT_1_COLON_X (DISPLAY_MIDDLE_X + ((40/6) * CLK_TEXT_SIZE)) //X location
for minutes to seconds colon
25#define TEXT_0_CURSOR_Y (DISPLAY_MIDDLE_Y - ((20/6) * CLK_TEXT_SIZE)) //Y location
for all text
26
27#define FAR_SCALER_X (105/6) // A scaler in the X direction for the farthest X point
(divide by 6 because 6 is the standard text size)
28#define FAR_SCALER_Y (90/6) // A scaler in the Y direction for the farthest Y point
(divide by 6 because 6 is the standard text size)
29#define MID_SCALER_X (30/6) // A scaler in the X direction for the mid X point (divide
by 6 because 6 is the standard text size)
30#define MID_SCALER_Y (40/6) // A scaler in the Y direction for the mid Y point (divide
by 6 because 6 is the standard text size)
31#define MID_SCALER_BASE_X (60/6) // A scaler in the X direction for the farthest X
point (divide by 6 because 6 is the standard text size)
32
33
34// TRIANGLE 0
35#define TRI_0_X0 (DISPLAY_MIDDLE_X - (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the top of the triangle
36#define TRI_0_Y0 (DISPLAY_MIDDLE_Y - (FAR_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the top of the triangle
37#define TRI_0_X1 (TRI_1_X1 - (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate
the X coordinate for the left side of the triangle
38#define TRI_0_Y1 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the left side of the triangle
39#define TRI_0_X2 (TRI_0_X1 + (MID_SCALER_BASE_X * CLK_TEXT_SIZE))
//Locate the X coordinate for the right side of the triangle
40#define TRI_0_Y2 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the right side of the triangle
41
42// TRIANGLE 1
43#define TRI_1_X0 (DISPLAY_MIDDLE_X) //Locate the X
```

clockDisplay.c

```
coordinate for the top of the triangle
44 #define TRI_1_Y0 (DISPLAY_MIDDLE_Y - (FAR_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the top of the triangle
45 #define TRI_1_X1 (DISPLAY_MIDDLE_X - (MID_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the left side of the triangle
46 #define TRI_1_Y1 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the left side of the triangle
47 #define TRI_1_X2 (DISPLAY_MIDDLE_X + (MID_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the right side of the triangle
48 #define TRI_1_Y2 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the right side of the triangle
49
50 // TRIANGLE 2
51 #define TRI_2_X0 (DISPLAY_MIDDLE_X + (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the top of the triangle
52 #define TRI_2_Y0 (DISPLAY_MIDDLE_Y - (FAR_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the top of the triangle
53 #define TRI_2_X1 (TRI_1_X1 + (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the left side of the triangle
54 #define TRI_2_Y1 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the left side of the triangle
55 #define TRI_2_X2 (TRI_2_X1 + (MID_SCALER_BASE_X * CLK_TEXT_SIZE)) //Locate
the X coordinate for the right side of the triangle
56 #define TRI_2_Y2 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the right side of the triangle
57
58 // TRIANGLE 3
59 #define TRI_3_X0 (DISPLAY_MIDDLE_X - (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the top of the triangle
60 #define TRI_3_Y0 (DISPLAY_MIDDLE_Y + (FAR_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the top of the triangle
61 #define TRI_3_X1 (TRI_1_X1 - (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the left side of the triangle
62 #define TRI_3_Y1 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the left side of the triangle
63 #define TRI_3_X2 (TRI_0_X1 + (MID_SCALER_BASE_X * CLK_TEXT_SIZE)) //Locate
the X coordinate for the right side of the triangle
64 #define TRI_3_Y2 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the right side of the triangle
65
66 // TRIANGLE 4
67 #define TRI_4_X0 (DISPLAY_MIDDLE_X) //Locate the X
coordinate for the top of the triangle
68 #define TRI_4_Y0 (DISPLAY_MIDDLE_Y + (FAR_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the top of the triangle
69 #define TRI_4_X1 (DISPLAY_MIDDLE_X - (MID_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the left side of the triangle
70 #define TRI_4_Y1 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the left side of the triangle
71 #define TRI_4_X2 (DISPLAY_MIDDLE_X + (MID_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the right side of the triangle
72 #define TRI_4_Y2 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the right side of the triangle
73
74 // TRIANGLE 5
75 #define TRI_5_X0 (DISPLAY_MIDDLE_X + (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the top of the triangle
76 #define TRI_5_Y0 (DISPLAY_MIDDLE_Y + (FAR_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
```

clockDisplay.c

```
coordinate for the top of the triangle
77 #define TRI_5_X1 (TRI_1_X1 + (FAR_SCALER_X * CLK_TEXT_SIZE)) //Locate the X
coordinate for the left side of the triangle
78 #define TRI_5_Y1 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the left side of the triangle
79 #define TRI_5_X2 (TRI_2_X1 + (MID_SCALER_BASE_X * CLK_TEXT_SIZE)) //Locate
the X coordinate for the right side of the triangle
80 #define TRI_5_Y2 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE)) //Locate the Y
coordinate for the right side of the triangle
81
82 //*****VAR_ FOR_TIME_KEEPING*****
83 #define HOURS_MAX 12 //MAX number that hours can be
84 #define MAX_MINUTES_AND_SECONDS 59 //MAX number minutes and seconds can be
85 #define MIN_TIME 0 //MIN number that time can be
86 #define NUM_OF_LOOPS 10 //number of time the for loops will
increment the clock
87 #define DELAY_X_TEN 100 //how long it take to tick when going 10x
as fast
88 #define DELAY_NORMAL 500 //how long it take to tick when going at a
normal speed
89 #define FALSE 0
90 #define TRUE 1
91 uint8_t hours = 12; //var to store and init the hours string
92 uint8_t minutes = 59; //var to store and init the minutes string
93 uint8_t seconds = 59; //var to store and init the seconds string
94 uint8_t updatedHours = 12; //var to store and init the updated hours string
95 uint8_t updatedMinutes = 59; //var to store and init the updated minutes string
96 uint8_t updatedSeconds = 59; //var to store and init the updated seconds string
97 char hourString[2]; // char to display the hours on the board
98 char minuteString[2]; // char to display the minutes on the board
99 char secondString[2]; // char to display the seconds on the board
100
101 void clockDisplay_init(){
102
103     display_init(); //initializes the display board
104     display_fillScreen(DISPLAY_BLACK); //displays the screen as black
105     display_setTextSize(CLK_TEXT_SIZE); //set the text height
106     display_setCursor(TEXT_0_CURSOR_X, TEXT_0_CURSOR_Y); //set the text cursor for
hours
107
108     display_setTextColor(DISPLAY_GREEN, DISPLAY_BLACK); //set the text color to
green and the background to black
109     display_setCursor(TEXT_0_COLON_X, TEXT_0_CURSOR_Y); //set the text to write a
colon between the hours and min
110     display_println(CLOCK_SEPERATOR); //set the colon between
hours and min
111     display_setCursor(TEXT_1_COLON_X, TEXT_0_CURSOR_Y); //set the text to write a
colon between the min and sec
112     display_println(CLOCK_SEPERATOR); //set the colon between
min and sec
113
114     display_fillTriangle(TRI_1_X0, TRI_1_Y0, TRI_1_X1, TRI_1_Y1, TRI_1_X2, TRI_1_Y2,
DISPLAY_GREEN); //create and fill triangle 0
115     display_fillTriangle(TRI_0_X0, TRI_0_Y0, TRI_0_X1, TRI_0_Y1, TRI_0_X2, TRI_0_Y2,
DISPLAY_GREEN); //create and fill triangle 1
116     display_fillTriangle(TRI_2_X0, TRI_2_Y0, TRI_2_X1, TRI_2_Y1, TRI_2_X2, TRI_2_Y2,
DISPLAY_GREEN); //create and fill triangle 2
117     display_fillTriangle(TRI_3_X0, TRI_3_Y0, TRI_3_X1, TRI_3_Y1, TRI_3_X2, TRI_3_Y2,
```

clockDisplay.c

```

    DISPLAY_GREEN); //create and fill triangle 3
118    display_fillTriangle(TRI_4_X0, TRI_4_Y0, TRI_4_X1, TRI_4_Y1, TRI_4_X2, TRI_4_Y2,
    DISPLAY_GREEN); //create and fill triangle 4
119    display_fillTriangle(TRI_5_X0, TRI_5_Y0, TRI_5_X1, TRI_5_Y1, TRI_5_X2, TRI_5_Y2,
    DISPLAY_GREEN); //create and fill triangle 5
120
121    clockDisplay_updateTimeDisplay(TRUE);
122
123 }
124
125 // Updates the time display with latest time, making sure to update only those digits
    that
126 // have changed since the last update.
127 // if forceUpdateAll is true, update all digits.
128 void clockDisplay_updateTimeDisplay(bool forceUpdateAll){
129     if((hours != updatedHours) || forceUpdateAll){
130         hours = updatedHours; // update var hours to new hours
131         sprintf(hourString, "%2d", hours); // save hours to a string
132         display_setCursor(TEXT_0_CURSOR_X, TEXT_0_CURSOR_Y); //set cursor to the hours
    coordinate
133         display_println(hourString); //print hours to the screen
134     }
135     if((minutes != updatedMinutes) || forceUpdateAll){
136         minutes = updatedMinutes; // update var hours to new hours
137         sprintf(minuteString, "%02d", minutes); // save hours to a string
138         display_setCursor(TEXT_1_CURSOR_X, TEXT_0_CURSOR_Y); //set cursor to the hours
    coordinate
139         display_println(minuteString); //print minutes to the screen
140     }
141     if((seconds != updatedSeconds) || forceUpdateAll){
142         seconds = updatedSeconds; // update var hours to new hours
143         sprintf(secondString, "%02d", seconds); // save hours to a string
144         display_setCursor(TEXT_2_CURSOR_X, TEXT_0_CURSOR_Y); //set cursor to the hours
    coordinate
145         display_println(secondString); //print seconds to the screen
146     }
147 }
148 // Reads the touched coordinates and performs the increment or decrement,
149 // depending upon the touched region.
150 void clockDisplay_performIncDec(){
151     int16_t x = 0; //Initialize x var (x location on the board)
152     int16_t y = 0; //Initialize y var (y location on the board)
153     uint8_t z = 0; //Initialize z var (z location on the board)
154     display_getTouchedPoint(&x, &y, &z); //inputs the coordinates of the screen touch
155     if (y < DISPLAY_MIDDLE_Y){ // check to see if the touch is in the top half of the
    screen
156         if(x < LINE_0_X){ //check if x is in 0 box
157             updatedHours += 1; // increment hours by one
158             if (updatedHours >= (HOURS_MAX + 1)){ // check if hours are above MAX
159                 updatedHours = ((MIN_TIME) + 1); // if hours are above MAX, reset it
    to 1
160             }
161         }
162         else if((x > LINE_0_X) && (x < LINE_1_X)){ //check if x is in 1 box
163             updatedMinutes += 1; // increment hours by one
164             if (updatedMinutes >= (MAX_MINUTES_AND_SECONDS + 1)){ // check if minutes
    are above MAX
165                 updatedMinutes = (MIN_TIME); // if minutes are above MAX, reset it to 0

```


clockDisplay.c

```

166     }
167 }
168     else if((x > LINE_1_X) && (x < DISPLAY_WIDTH)){ //check if x is in 2 box
169         updatedSeconds += 1; // increment hours by one
170         if (updatedSeconds >= (MAX_MINUTES_AND_SECONDS + 1)){ // check if seconds
are above MAX
171             updatedSeconds = (MIN_TIME); // if seconds are above MAX, reset it to
0
172         }
173     }
174     else{
175         printf("%s", "ERROR invalid board touch"); // print out error if it touch
is invalid
176     }
177 }
178     else if(y > DISPLAY_MIDDLE_Y){ // check to see if the touch is in the bottom half
of the screen
179         if(x < LINE_0_X){ //check if x is in 3 box
180             updatedHours -= 1; // decrement hours by one
181             if (updatedHours == MIN_TIME){ // check if hours are below MIN time
182                 updatedHours = HOURS_MAX; // if hours are below, reset it to max hours
183             }
184         }
185         else if((x > LINE_0_X) && (x < LINE_1_X)){ //check if x is in 4 box
186             if (updatedMinutes == MIN_TIME){ // check if minutes are below MIN time
187                 updatedMinutes = MAX_MINUTES_AND_SECONDS; // if minutes are below,
reset it to max minutes
188             }
189             else{
190                 updatedMinutes -= 1; // decrement minutes by one
191             }
192         }
193         else if((x > LINE_1_X) && (x < DISPLAY_WIDTH)){ //check if x is in 5 box
194             if (updatedSeconds == MIN_TIME){ // check if seconds are below MIN time
195                 updatedSeconds = MAX_MINUTES_AND_SECONDS; // if seconds are below,
reset it to max second
196             }
197             else{
198                 updatedSeconds -= 1; // decrement second by one
199             }
200         }
201         else{
202             printf("%s", "ERROR invalid board touch"); // print out error statement if
the touch isn't valid
203         }
204     }
205     else{
206         printf("%s", "ERROR invalid board touch"); // print out error statement if
the touch isn't valid
207     }
208     clockDisplay_updateTimeDisplay(FALSE);
209 }
210
211 // Advances the time forward by 1 second and update the display.
212 void clockDisplay_advanceTimeOneSecond(){
213     updatedSeconds += 1; // advance seconds 1
214     if (updatedSeconds >= MAX_MINUTES_AND_SECONDS + 1){ // check if second is equal
to or greater than 59

```

clockDisplay.c

```
215         updatedSeconds = MIN_TIME; //set the seconds back to 0
216         updatedMinutes += 1; // advance minutes 1
217     }
218     if (updatedMinutes >= MAX_MINUTES_AND_SECONDS + 1){ // check if minutes is equal
to or greater than 59
219         updatedMinutes = MIN_TIME; //set the minutes back to 0
220         updatedHours += 1; // advance hours 1
221     }
222     if (updatedHours >= HOURS_MAX + 1){ // check if hours is equal to or greater than
13
223         updatedHours = ((MIN_TIME) + 1); //set the seconds back to 1 (because hours
can be 0)
224     }
225 };
226
227 // Run a test of clock-display functions.
228 void clockDisplay_runTest(){
229     clockDisplay_init(); // Initialize the clock
230     clockDisplay_updateTimeDisplay(TRUE); //update the display to 12:59:59
231     for (int i = 0; i < NUM_OF_LOOPS; i++){ //decrement hours minutes and seconds 10
times
232         updatedHours -= 1; // decrement hours
233         updatedMinutes -= 1; // decrement minutes
234         updatedSeconds -= 1; // decrement second
235         clockDisplay_updateTimeDisplay(FALSE); //update the display
236         utils_msDelay(DELAY_NORMAL); // wait 500 m/s
237     }
238     for (int i = 0; i < NUM_OF_LOOPS; i++){ //increment hours minutes and seconds 10
times
239         updatedHours += 1; // increment hours
240         updatedMinutes += 1; // increment minutes
241         updatedSeconds += 1; // increment second
242         clockDisplay_updateTimeDisplay(FALSE); //update the display
243         utils_msDelay(DELAY_NORMAL); // wait 500 m/s
244     }
245     for (int i = 0; i < (NUM_OF_LOOPS*NUM_OF_LOOPS); i++){
246         clockDisplay_advanceTimeOneSecond(); //increment every tick
247         clockDisplay_updateTimeDisplay(FALSE); // update the display
248         utils_msDelay(DELAY_X_TEN); //increment at x10
249     }
250 };
251
```

clockControl.h

```
2  * clockControl.h
7
8  #ifndef CLOCKCONTROL_H_
9  #define CLOCKCONTROL_H_
10
11 // Standard tick function.
12 void clockControl_tick();
13
14 // Call this before you call clockControl_tick().
15 void clockControl_init();
16
17
18 #endif /* CLOCKCONTROL_H_ */
19
```

clockControl.c

```
2 * clockControl.c
7
8 #include "clockControl.h"
9 #include "clockDisplay.h"
10 #include "supportFiles/display.h"
11 #include <stdio.h>
12
13 #define ADC_COUNTER_MAX_VALUE 1 //the value of when the counter can flip when
    triggered by a touch
14 #define AUTO_COUNTER_MAX_VALUE 5 //the value of when the counter can flip
    automatically
15 #define RATE_COUNTER_MAX_VALUE 1 //the rate that the counter increments at
16 #define TOUCH_EXPIRED 10 //how many ticks it take for the touch to expire and
    exit the state
17 #define INITIALIZE_VAR 0 //value to initialize most var's
18 #define TRUE 1
19 #define FALSE 0
20
21 int8_t adcCounter = INITIALIZE_VAR; //counter for the touch input
22 int16_t autoCounter = INITIALIZE_VAR; //counter for the automatic increment
23 int16_t rateCounter = INITIALIZE_VAR; //counter for how fast it increments
24 int8_t touched = INITIALIZE_VAR; //var to store if the screen has been
    touched
25 int8_t soak = INITIALIZE_VAR;
26
27
28
29 // States for the controller state machine.
30 enum clockControl_st_t {
31     init_st, // Start here, stay in this state for just one tick.
32     never_touched_st, // Wait here until the first touch - clock is disabled
    until set.
33     waiting_for_touch_st, // waiting for touch, clock is enabled and running.
34     ad_timer_running_st, // waiting for the touch-controller ADC to settle.
35     auto_timer_running_st, // waiting for the auto-update delay to expire
    // (user is holding down button for auto-inc/dec)
36     rate_timer_running_st, // waiting for the rate-timer to expire to know when to
    perform the auto inc/dec.
37     rate_timer_expired_st, // when the rate-timer expires, perform the inc/dec
    function.
38     add_second_to_clock_st // add a second to the clock time and reset the ms
    counter.
39 } currentState = init_st;
40
41
42
43 // This is a debug state print routine. It will print the names of the states each
44 // time tick() is called. It only prints states if they are different than the
45 // previous state.
46 void debugStatePrint() {
47     static clockControl_st_t previousState;
48     static bool firstPass = true;
49     // Only print the message if:
50     // 1. This the first pass and the value for previousState is unknown.
51     // 2. previousState != currentState - this prevents reprinting the same state name
    over and over.
52     if (previousState != currentState || firstPass) {
53         firstPass = false; // previousState will be defined, firstPass is
    false.
54     }
```

clockControl.c

```

54     previousState = currentState;    // keep track of the last state that you were
in.
55     //printf("msCounter:%d\n\r", msCounter);
56     switch(currentState) {           // This prints messages based upon the state
that you were in.
57         case init_st:
58             printf("init_st\n\r");
59             break;                    //exit state
60         case never_touched_st:
61             printf("never_touched_st\n\r");
62             break;                    //exit state
63         case waiting_for_touch_st:
64             printf("waiting_for_touch_st\n\r");
65             break;                    //exit state
66         case ad_timer_running_st:
67             printf("ad_timer_running_st\n\r");
68             break;                    //exit state
69         case auto_timer_running_st:
70             printf("auto_timer_running_st\n\r");
71             break;                    //exit state
72         case rate_timer_running_st:
73             printf("rate_timer_running_st\n\r");
74             break;                    //exit state
75         case rate_timer_expired_st:
76             printf("rate_timer_expired_st\n\r");
77             break;                    //exit state
78         case add_second_to_clock_st:
79             break;                    //exit state
80     }
81 }
82 }
83
84 void clockControl_init() {
85     currentState = init_st; //Initializes the state machine to the initial state
86 }
87
88 void clockControl_tick() {
89     debugStatePrint(); //print out the debug function as a state is entered
90     switch(currentState) { //moore output switch
91         case init_st: //empty (didn't use this state)
92             break; //exit state
93         case never_touched_st: //empty (didn't use this state)
94             break; //exit state
95         case waiting_for_touch_st: //name of state
96             adcCounter = INITIALIZE_VAR; //Initialize the counter var's
97             autoCounter = INITIALIZE_VAR; //Initialize the counter var's
98             rateCounter = INITIALIZE_VAR; //Initialize the counter var's
99             if (touched){// if the display has been touched
100                 if (soak == TOUCH_EXPIRED){// if soak has incremented (waited long enough)
the refresh screen:
101                     clockDisplay_advanceTimeOneSecond();// increment time by one second
102                     clockDisplay_updateTimeDisplay(0); // update the display with the new
time
103                     soak = INITIALIZE_VAR; // reset the counter
104                 }
105                 else {
106                     soak++;// else, add one to the counter
107                 }

```

clockControl.c

```

108     }
109     break; //exit state
110     case ad_timer_running_st: //name of state
111         if (!touched){
112             touched = TRUE; // set to 1
113         }
114         adcCounter++; // Increment the counter
115         break; //exit state
116     case auto_timer_running_st: //name of state
117         autoCounter++; // Increment the counter
118         break; //exit state
119     case rate_timer_running_st: //name of state
120         rateCounter++; // Increment the timer.
121         break; //exit state
122     case rate_timer_expired_st: //name of state
123         rateCounter = INITIALIZE_VAR; // reset rateTimer
124         break; //exit state
125     default:
126         printf("clockControl_tick state action: hit default\n\r");
127         break; //exit state
128 }
129
130 // Perform state update, transition
131 switch(currentState) {
132     case init_st: //name of state
133         currentState = waiting_for_touch_st; // Initialize state machine
134         break; //exit state
135     case never_touched_st: //name of state
136         //empty
137         break; //exit state
138     case waiting_for_touch_st: //name of state
139         if(display_isTouched()){ //check if the screen is touched
140             display_clearOldTouchData(); // clear old touch data
141             currentState = ad_timer_running_st; // go to the next state
142         }
143         break; //exit state
144     case ad_timer_running_st: //name of state
145         if (display_isTouched() && adcCounter == ADC_COUNTER_MAX_VALUE){ //check if
the screen is touched
146             currentState = auto_timer_running_st; // go to the next state
147         }
148         else if (!display_isTouched() && adcCounter == ADC_COUNTER_MAX_VALUE){
149             clockDisplay_performIncDec();
150             currentState = waiting_for_touch_st; // go to the next state
151         }
152         break; //exit state
153     case auto_timer_running_st: //name of state
154         if (display_isTouched() && autoCounter == AUTO_COUNTER_MAX_VALUE){ //check if
the screen is touched
155             currentState = rate_timer_running_st; // go to the next state
156         }
157         else if (!display_isTouched()){
158             clockDisplay_performIncDec();
159             currentState = waiting_for_touch_st; // go to the next state
160         }
161         break; //exit state
162     case rate_timer_running_st: //name of state
163         if (display_isTouched() && rateCounter == RATE_COUNTER_MAX_VALUE){ //check if

```

clockControl.c

```
the screen is touched
164         currentState = rate_timer_expired_st; // go to the next state
165     }
166     else if (!display_isTouched()){
167         currentState = waiting_for_touch_st; // go to the next state
168     }
169     break; //exit state
170     case rate_timer_expired_st: //name of state
171         if (display_isTouched()){ //check if the screen is touched
172             clockDisplay_performIncDec();
173             currentState = rate_timer_running_st; // go to the next state
174         }
175         else if (!display_isTouched()){
176             currentState = waiting_for_touch_st; // go to the next state
177         }
178         break; //exit state
179     case add_second_to_clock_st: //empty
180         break; //exit state
181     default:
182         printf("clockControl_tick state update: hit default\n\r");
183         break; //exit state
184 }
185 }
186
187
188
189
190
```