```
 * clockMain.c

/***********************************
************ Flag Method ***********
************************************/
#include <stdio.h>
#include "supportFiles/leds.h"
#include "supportFiles/globalTimer.h"
#include "supportFiles/interrupts.h"
#include "supportFiles/utils.h"
#include <stdbool.h>
#include <stdint.h>
#include "clockControl.h"
#include "clockDisplay.h"
#include "supportFiles/display.h"

#include "xparameters.h"

#define TOTAL_SECONDS 60
// The formula for computing the load value is based upon the formula from 4.1.1 (calculating timer intervals)
// in the Cortex-A9 MPCore Technical Reference Manual 4-2.
// Assuming that the prescaler = 0, the formula for computing the load value based upon the desired period is:
// load-value = (period * timer-clock) - 1
#define TIMER_PERIOD 91.0E-3 // You can change this value to a value that you select.10.0E-3
#define TIMER_CLOCK_FREQUENCY (XPAR_CPU_CORTEXA9_0_CPU_CLK_FREQ_HZ / 2)
#define TIMER_LOAD_VALUE ((TIMER_PERIOD * TIMER_CLOCK_FREQUENCY) - 1.0)

/*
int main() {
        clockDisplay_init();
        clockControl_init();
    while (1) {
        clockControl_tick();
        utils_msDelay(100);
    }
}
*/
/*
int main()
{
    // Initialize the GPIO LED driver and print out an error message if it fails (argument = true).
       // You need to init the LEDs so that LD4 can function as a heartbeat.
    leds_init(true);
    // Init all interrupts (but does not enable the interrupts at the devices).
    // Prints an error message if an internal failure occurs because the argument = true.
    interrupts_initAll(true);
    interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
    u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
    printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
    // Allow the timer to generate interrupts.
    interrupts_enableTimerGlobalInts();
    // Initialization of the clock display is not time-dependent, do it outside of the state machine.
```

```c
    clockDisplay_init();
    // Keep track of your personal interrupt count. Want to make sure that you don't miss any interrupts.
     int32_t personalInterruptCount = 0;
    // Start the private ARM timer running.
    interrupts_startArmPrivateTimer();
    // Enable interrupts at the ARM.
    interrupts_enableArmInts();
    // interrupts_isrInvocationCount() returns the number of times that the timer ISR was invoked.
    // This value is maintained by the timer ISR. Compare this number with your own local
    // interrupt count to determine if you have missed any interrupts.
     while (interrupts_isrInvocationCount() < (TOTAL_SECONDS * privateTimerTicksPerSecond)) {
      if (interrupts_isrFlagGlobal) {  // This is a global flag that is set by the timer interrupt handler.
          // Count ticks.
        personalInterruptCount++;
        clockControl_tick();
          interrupts_isrFlagGlobal = 0;
      }
   }
   interrupts_disableArmInts();
   printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
   printf("internal interrupt count: %ld\n\r", personalInterruptCount);
   return 0;
}
///

void isr_function() {

  // Leave blank for the flag method.
  // I already set the flag for you in another routine.
}
*/


static uint32_t isr_functionCallCount = 0;

int main()
{
    // Initialize the GPIO LED driver and print out an error message if it fails (argument = true).
    // You need to init the LEDs so that LD4 can function as a heartbeat.
    leds_init(true);
    // Init all interrupts (but does not enable the interrupts at the devices).
    // Prints an error message if an internal failure occurs because the argument = true.
    interrupts_initAll(true);
    interrupts_setPrivateTimerLoadValue(TIMER_LOAD_VALUE);
    printf("timer load value:%ld\n\r", (int32_t) TIMER_LOAD_VALUE);
    u32 privateTimerTicksPerSecond = interrupts_getPrivateTimerTicksPerSecond();
    printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);
    interrupts_enableTimerGlobalInts();
    // Initialization of the clock display is not time-dependent, do it outside of the state machine.
    clockDisplay_init();
    // Start the private ARM timer running.
    interrupts_startArmPrivateTimer();
    // Enable interrupts at the ARM.
    interrupts_enableArmInts();
    // The while-loop just waits until the total number of timer ticks have occurred before proceeding.
    while (interrupts_isrInvocationCount() < (TOTAL_SECONDS * privateTimerTicksPerSecond));
    // All done, now disable interrupts and print out the interrupt counts.
    interrupts_disableArmInts();
    printf("isr invocation count: %ld\n\r", interrupts_isrInvocationCount());
    printf("internal interrupt count: %ld\n\r", isr_functionCallCount);
    return 0;
}

// The clockControl_tick() function is now called directly by the timer interrupt service routine.
void isr_function() {
    clockControl_tick();
   isr_functionCallCount++;
    // Add the necessary code here.
}


```

```
 * clockDisplay.h

#ifndef CLOCKDISPLAY_H_
#define CLOCKDISPLAY_H_

#include <stdbool.h>

// Called only once - performs any necessary inits.
// This is a good place to draw the triangles and any other
// parts of the clock display that will never change.
void clockDisplay_init();

// Updates the time display with latest time, making sure to update only those digits that
// have changed since the last update.
// if forceUpdateAll is true, update all digits.
void clockDisplay_updateTimeDisplay(bool forceUpdateAll);

// Reads the touched coordinates and performs the increment or decrement,
// depending upon the touched region.
void clockDisplay_performIncDec();

// Advances the time forward by 1 second and update the display.
void clockDisplay_advanceTimeOneSecond();

// Run a test of clock-display functions.
void clockDisplay_runTest();


#endif /* CLOCKDISPLAY_H_ */

```

```
 * clockDisplay.c
#include "clockDisplay.h" //include the header file for this .c file
#include "supportFiles/display.h" //include the display header so you can display to the board
#include "supportFiles/utils.h" //this includes a file that makes the clock work
#include "stdio.h"
//#include <stdio.h>

#define CLK_TEXT_SIZE 6 // set the text size to 6
#define CLOCK_SEPERATOR ":" //is the colon to separate the hours min and sec
#define DISPLAY_MIDDLE_X (DISPLAY_WIDTH / 2) //finds the middle of the board in the X direction 160 pixels
#define DISPLAY_MIDDLE_Y (DISPLAY_HEIGHT / 2) //finds the middle of the board in the Y direction 120 pixels
#define LINE_0_X (DISPLAY_WIDTH / 3)
#define LINE_1_X (2 * (DISPLAY_WIDTH / 3))

#define TEXT_0_CURSOR_X (DISPLAY_MIDDLE_X - ((135/6) * CLK_TEXT_SIZE))   //X location for hours
#define TEXT_1_CURSOR_X (DISPLAY_MIDDLE_X - ((30/6) * CLK_TEXT_SIZE))      //X location for minutes
#define TEXT_2_CURSOR_X (DISPLAY_MIDDLE_X + ((75/6) * CLK_TEXT_SIZE))   //X location for seconds
#define TEXT_0_COLON_X (DISPLAY_MIDDLE_X - ((70/6) * CLK_TEXT_SIZE))   //X location for hours to minutes colon 65
#define TEXT_1_COLON_X (DISPLAY_MIDDLE_X + ((40/6) * CLK_TEXT_SIZE))    //X location for minutes to seconds colon
#define TEXT_0_CURSOR_Y (DISPLAY_MIDDLE_Y - ((20/6) * CLK_TEXT_SIZE))   //Y location for all text

#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)
#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)
#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)
#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)
#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)


// TRIANGLE 0
#define TRI_0_X0 (DISPLAY_MIDDLE_X - (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the top of the triangle
#define TRI_0_Y0 (DISPLAY_MIDDLE_Y - (FAR_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the top of the triangle
#define TRI_0_X1 (TRI_1_X1 -         (FAR_SCALER_X * CLK_TEXT_SIZE))       //Locate the X coordinate for the left side of the triangle
#define TRI_0_Y1 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the left side of the triangle
#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
#define TRI_0_Y2 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the right side of the triangle

// TRIANGLE 1
#define TRI_1_X0 (DISPLAY_MIDDLE_X)                               //Locate the X
```

```
coordinate for the top of the triangle
#define TRI_1_Y0 (DISPLAY_MIDDLE_Y - (FAR_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the top of the triangle
#define TRI_1_X1 (DISPLAY_MIDDLE_X - (MID_SCALER_X * CLK_TEXT_SIZE))    //Locate the X coordinate for the left side of the triangle
#define TRI_1_Y1 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the left side of the triangle
#define TRI_1_X2 (DISPLAY_MIDDLE_X + (MID_SCALER_X * CLK_TEXT_SIZE))    //Locate the X coordinate for the right side of the triangle
#define TRI_1_Y2 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the right side of the triangle

// TRIANGLE 2
#define TRI_2_X0 (DISPLAY_MIDDLE_X + (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the top of the triangle
#define TRI_2_Y0 (DISPLAY_MIDDLE_Y - (FAR_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the top of the triangle
#define TRI_2_X1 (TRI_1_X1 +         (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the left side of the triangle
#define TRI_2_Y1 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the left side of the triangle
#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
#define TRI_2_Y2 (DISPLAY_MIDDLE_Y - (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the right side of the triangle

// TRIANGLE 3
#define TRI_3_X0 (DISPLAY_MIDDLE_X - (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the top of the triangle
#define TRI_3_Y0 (DISPLAY_MIDDLE_Y + (FAR_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the top of the triangle
#define TRI_3_X1 (TRI_1_X1 -         (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the left side of the triangle
#define TRI_3_Y1 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the left side of the triangle
#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
#define TRI_3_Y2 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the right side of the triangle

// TRIANGLE 4
#define TRI_4_X0 (DISPLAY_MIDDLE_X)                                 //Locate the X coordinate for the top of the triangle
#define TRI_4_Y0 (DISPLAY_MIDDLE_Y + (FAR_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the top of the triangle
#define TRI_4_X1 (DISPLAY_MIDDLE_X - (MID_SCALER_X * CLK_TEXT_SIZE))    //Locate the X coordinate for the left side of the triangle
#define TRI_4_Y1 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the left side of the triangle
#define TRI_4_X2 (DISPLAY_MIDDLE_X + (MID_SCALER_X * CLK_TEXT_SIZE))    //Locate the X coordinate for the right side of the triangle
#define TRI_4_Y2 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the right side of the triangle

//TRIANGLE 5
#define TRI_5_X0 (DISPLAY_MIDDLE_X + (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the top of the triangle
#define TRI_5_Y0 (DISPLAY_MIDDLE_Y + (FAR_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y
```

```
coordinate for the top of the triangle
#define TRI_5_X1 (TRI_1_X1 +         (FAR_SCALER_X * CLK_TEXT_SIZE))   //Locate the X coordinate for the left side of the triangle
#define TRI_5_Y1 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the left side of the triangle
#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
#define TRI_5_Y2 (DISPLAY_MIDDLE_Y + (MID_SCALER_Y * CLK_TEXT_SIZE))    //Locate the Y coordinate for the right side of the triangle

//*********VAR_ FOR_TIME_KEEPING*********
#define HOURS_MAX 12                            //MAX number that hours can be
#define MAX_MINUTES_AND_SECONDS 59              //MAX number minutes and seconds can be
#define MIN_TIME 0                              //MIN number that time can be
#define NUM_OF_LOOPS 10                         //number of time the for loops will increment the clock
#define DELAY_X_TEN 100                         //how long it take to tick when going 10x as fast
#define DELAY_NORMAL 500                        //how long it take to tick when going at a normal speed
#define FALSE 0
#define TRUE 1
    uint8_t hours = 12; //var to store and init the hours string
    uint8_t minutes = 59; //var to store and init the minutes string
    uint8_t seconds = 59; //var to store and init the seconds string
    uint8_t updatedHours = 12; //var to store and init the  updated hours string
    uint8_t updatedMinutes = 59; //var to store and init the updated minutes string
    uint8_t updatedSeconds = 59; //var to store and init the  updated seconds string
    char hourString[2];          // char to display the hours on the board
    char minuteString[2];        // char to display the minutes on the board
    char secondString[2];        // char to display the seconds on the board

void clockDisplay_init(){

    display_init(); //initializes the display board
    display_fillScreen(DISPLAY_BLACK); //displays the screen as black
    display_setTextSize(CLK_TEXT_SIZE);   //set the text height
    display_setCursor(TEXT_0_CURSOR_X, TEXT_0_CURSOR_Y);    //set the text cursor for hours

    display_setTextColor(DISPLAY_GREEN, DISPLAY_BLACK);     //set the text color to green and the background to black
    display_setCursor(TEXT_0_COLON_X, TEXT_0_CURSOR_Y);     //set the text to write a colon between the hours and min
    display_println(CLOCK_SEPERATOR);                       //set the colon between hours and min
    display_setCursor(TEXT_1_COLON_X, TEXT_0_CURSOR_Y);     //set the text to write a colon between the min and sec
    display_println(CLOCK_SEPERATOR);                       //set the colon between min and sec

    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
    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
    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
    display_fillTriangle(TRI_3_X0, TRI_3_Y0, TRI_3_X1, TRI_3_Y1, TRI_3_X2, TRI_3_Y2,
DISPLAY_GREEN); //create and fill triangle 3
    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
    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

    clockDisplay_updateTimeDisplay(TRUE);

}

// Updates the time display with latest time, making sure to update only those digits that
// have changed since the last update.
// if forceUpdateAll is true, update all digits.
void clockDisplay_updateTimeDisplay(bool forceUpdateAll){
    if((hours != updatedHours) || forceUpdateAll){
        hours = updatedHours; // update var hours to new hours
        sprintf(hourString, "%2d", hours); // save hours to a string
        display_setCursor(TEXT_0_CURSOR_X, TEXT_0_CURSOR_Y); //set cursor to the hours coordinate
        display_println(hourString); //print hours to the screen
    }
    if((minutes != updatedMinutes) || forceUpdateAll){
        minutes = updatedMinutes; // update var hours to new hours
        sprintf(minuteString, "%02d", minutes); // save hours to a string
        display_setCursor(TEXT_1_CURSOR_X, TEXT_0_CURSOR_Y); //set cursor to the hours coordinate
        display_println(minuteString); //print minutes to the screen
    }
    if((seconds != updatedSeconds) || forceUpdateAll){
        seconds = updatedSeconds; // update var hours to new hours
        sprintf(secondString, "%02d", seconds); // save hours to a string
        display_setCursor(TEXT_2_CURSOR_X, TEXT_0_CURSOR_Y); //set cursor to the hours coordinate
        display_println(secondString); //print seconds to the screen
    }
}
// Reads the touched coordinates and performs the increment or decrement,
// depending upon the touched region.
void clockDisplay_performIncDec(){
    int16_t x = 0; //Initialize x var (x location on the board)
    int16_t y = 0; //Initialize y var (y location on the board)
    uint8_t z = 0; //Initialize z var (z location on the board)
    display_getTouchedPoint(&x, &y, &z); //inputs the coordinates of the screen touch
    if (y < DISPLAY_MIDDLE_Y){ // check to see if the touch is in the top half of the screen
        if(x < LINE_0_X){ //check if x is in 0 box
            updatedHours += 1;// increment hours by one
            if (updatedHours >= (HOURS_MAX + 1)){ // check if hours are above MAX
                updatedHours = ((MIN_TIME) + 1); // if hours are above MAX, reset it to 1
            }
        }
        else if((x > LINE_0_X) && (x < LINE_1_X)){ //check if x is in 1 box
            updatedMinutes += 1;// increment hours by one
            if (updatedMinutes >= (MAX_MINUTES_AND_SECONDS + 1)){ // check if minutes are above MAX
                updatedMinutes = (MIN_TIME);// if minutes are above MAX, reset it to 0
            }
        }
        else if((x > LINE_1_X) && (x < DISPLAY_WIDTH)){ //check if x is in 2 box
            updatedSeconds += 1;// increment hours by one
            if (updatedSeconds >= (MAX_MINUTES_AND_SECONDS + 1)){ // check if seconds are above MAX
                updatedSeconds = (MIN_TIME); // if seconds are above MAX, reset it to 0
            }
        }
        else{
            printf("%s", "ERROR invalid board touch"); // print out error if it touch is invalid
        }
    }
    else if(y > DISPLAY_MIDDLE_Y){  // check to see if the touch is in the bottom half of the screen
        if(x < LINE_0_X){ //check if x is in 3 box
            updatedHours -= 1; // decrement hours by one
            if (updatedHours == MIN_TIME){ // check if hours are below MIN time
                updatedHours = HOURS_MAX; // if hours are below, reset it to max hours
            }
        }
        else if((x > LINE_0_X) && (x < LINE_1_X)){ //check if x is in 4 box
            if (updatedMinutes == MIN_TIME){  // check if minutes are below MIN time
                updatedMinutes = MAX_MINUTES_AND_SECONDS;  // if minutes are below, reset it to max minutes
            }
            else{
                updatedMinutes -= 1; // decrement minutes by one
            }
        }
        else if((x > LINE_1_X) && (x < DISPLAY_WIDTH)){ //check if x is in 5 box
            if (updatedSeconds == MIN_TIME){  // check if seconds are below MIN time
                updatedSeconds = MAX_MINUTES_AND_SECONDS;  // if seconds are below, reset it to max second
            }
            else{
                updatedSeconds -= 1; // decrement second by one
            }
        }
        else{
            printf("%s", "ERROR invalid board touch"); // print out error statement if the touch isn't valid
        }
    }
    else{
        printf("%s", "ERROR invalid board touch");  // print out error statement if the touch isn't valid
    }
    clockDisplay_updateTimeDisplay(FALSE);
}

// Advances the time forward by 1 second and update the display.
void clockDisplay_advanceTimeOneSecond(){
    updatedSeconds += 1; // advance seconds 1
    if (updatedSeconds >= MAX_MINUTES_AND_SECONDS + 1){  // check if second is equal to or greater than 59
```

```c
        updatedSeconds = MIN_TIME; //set the seconds back to 0
        updatedMinutes += 1;// advance minutes 1
    }
    if (updatedMinutes >= MAX_MINUTES_AND_SECONDS + 1){  // check if minutes is equal to or greater than 59
        updatedMinutes = MIN_TIME; //set the minutes back to 0
        updatedHours += 1; // advance hours 1
    }
    if (updatedHours >= HOURS_MAX + 1){ // check if hours is equal to or greater than 13
        updatedHours = ((MIN_TIME) + 1); //set the seconds back to 1 (because hours can be 0)
    }
};

// Run a test of clock-display functions.
void clockDisplay_runTest(){
    clockDisplay_init(); // Initialize the clock
    clockDisplay_updateTimeDisplay(TRUE); //update the display to 12:59:59
    for (int i = 0; i < NUM_OF_LOOPS; i++){ //decrement hours minutes and seconds 10 times
        updatedHours -= 1; // decrement hours
        updatedMinutes -= 1; // decrement minutes
        updatedSeconds -= 1; // decrement second
        clockDisplay_updateTimeDisplay(FALSE); //update the display
        utils_msDelay(DELAY_NORMAL);// wait 500 m/s
    }
    for (int i = 0; i < NUM_OF_LOOPS; i++){//increment hours minutes and seconds 10 times
        updatedHours += 1; // increment hours
        updatedMinutes += 1; // increment minutes
        updatedSeconds += 1; // increment second
        clockDisplay_updateTimeDisplay(FALSE); //update the display
        utils_msDelay(DELAY_NORMAL); // wait 500 m/s
    }
    for (int i = 0; i < (NUM_OF_LOOPS*NUM_OF_LOOPS); i++){
        clockDisplay_advanceTimeOneSecond(); //increment every tick
        clockDisplay_updateTimeDisplay(FALSE); // update the display
        utils_msDelay(DELAY_X_TEN); //increment at x10
    }
};

```

```
 * clockControl.h

#ifndef CLOCKCONTROL_H_
#define CLOCKCONTROL_H_

// Standard tick function.
void clockControl_tick();

// Call this before you call clockControl_tick().
void clockControl_init();


#endif /* CLOCKCONTROL_H_ */

```

```
 * clockControl.c

#include "clockControl.h"
#include "clockDisplay.h"
#include "supportFiles/display.h"
#include <stdio.h>

#define ADC_COUNTER_MAX_VALUE 1  //the value of when the counter can flip when triggered by a touch
#define AUTO_COUNTER_MAX_VALUE 5 //the value of when the counter can flip automatically
#define RATE_COUNTER_MAX_VALUE 1 //the rate that the counter increments at
#define TOUCH_EXPIRED 10         //how many ticks it take for the touch to expire and exit the state
#define INITIALIZE_VAR 0         //value to initialize most var's
#define TRUE 1
#define FALSE 0

int8_t adcCounter = INITIALIZE_VAR;           //counter for the touch input
int16_t autoCounter = INITIALIZE_VAR;         //counter for the automatic increment
int16_t rateCounter = INITIALIZE_VAR;         //counter for how fast it increments
int8_t touched = INITIALIZE_VAR;              //var to store if the screen has been touched
int8_t soak = INITIALIZE_VAR;



// States for the controller state machine.
enum clockControl_st_t {
    init_st,                 // Start here, stay in this state for just one tick.
    never_touched_st,        // Wait here until the first touch - clock is disabled until set.
    waiting_for_touch_st,    // waiting for touch, clock is enabled and running.
    ad_timer_running_st,     // waiting for the touch-controller ADC to settle.
    auto_timer_running_st,   // waiting for the auto-update delay to expire
                                 // (user is holding down button for auto-inc/dec)
    rate_timer_running_st,   // waiting for the rate-timer to expire to know when to perform the auto inc/dec.
    rate_timer_expired_st,   // when the rate-timer expires, perform the inc/dec function.
    add_second_to_clock_st   // add a second to the clock time and reset the ms counter.
} currentState = init_st;


// This is a debug state print routine. It will print the names of the states each
// time tick() is called. It only prints states if they are different than the
// previous state.
void debugStatePrint() {
  static clockControl_st_t previousState;
  static bool firstPass = true;
  // Only print the message if:
  // 1. This the first pass and the value for previousState is unknown.
  // 2. previousState != currentState - this prevents reprinting the same state name over and over.
  if (previousState != currentState || firstPass) {
    firstPass = false;                // previousState will be defined, firstPass is false.
    previousState = currentState;     // keep track of the last state that you were in.
    //printf("msCounter:%d\n\r", msCounter);
    switch(currentState) {            // This prints messages based upon the state that you were in.
      case init_st:
        printf("init_st\n\r");
        break;                      //exit state
      case never_touched_st:
        printf("never_touched_st\n\r");
        break;                      //exit state
      case waiting_for_touch_st:
        printf("waiting_for_touch_st\n\r");
        break;                      //exit state
      case ad_timer_running_st:
        printf("ad_timer_running_st\n\r");
        break;                      //exit state
      case auto_timer_running_st:
        printf("auto_timer_running_st\n\r");
        break;                      //exit state
      case rate_timer_running_st:
        printf("rate_timer_running_st\n\r");
        break;                      //exit state
      case rate_timer_expired_st:
        printf("rate_timer_expired_st\n\r");
        break;                      //exit state
      case add_second_to_clock_st:
          break;                    //exit state
     }
  }
}

void clockControl_init() {
  currentState = init_st; //Initializes the state machine to the initial state
}

void clockControl_tick() {
    debugStatePrint(); //print out the debug function as a state is entered
  switch(currentState) { //moore output switch
    case init_st: //empty (didn't use this state)
      break; //exit state
    case never_touched_st: //empty (didn't use this state)
      break; //exit state
    case waiting_for_touch_st: //name of state
        adcCounter = INITIALIZE_VAR;   //Initialize the counter var's
        autoCounter = INITIALIZE_VAR;  //Initialize the counter var's
        rateCounter = INITIALIZE_VAR;  //Initialize the counter var's
        if (touched){// if the display has been touched
            if (soak == TOUCH_EXPIRED){// if soak has incremented (waited long enough) the refresh screen:
                clockDisplay_advanceTimeOneSecond();// increment time by one second
                clockDisplay_updateTimeDisplay(0);  // update the display with the new time
                soak = INITIALIZE_VAR; // reset the counter
            }
            else {
                soak++;// else, add one to the counter
            }
        }
      break;//exit state
    case ad_timer_running_st:    //name of state
        if (!touched){
          touched = TRUE;       // set to 1
        }
        adcCounter++;           // Increment the counter
      break;//exit state
    case auto_timer_running_st: //name of state
        autoCounter++;          // Increment the counter
      break;//exit state
    case rate_timer_running_st: //name of state
        rateCounter++;          // Increment the timer.
      break;//exit state
    case rate_timer_expired_st: //name of state
        rateCounter = INITIALIZE_VAR;// reset rateTimer
      break;//exit state
     default:
      printf("clockControl_tick state action: hit default\n\r");
      break;//exit state
  }

  // Perform state update, transition
  switch(currentState) {
    case init_st:    //name of state
        currentState = waiting_for_touch_st; // Initialize state machine
      break;//exit state
    case never_touched_st: //name of state
        //empty
      break;//exit state
    case waiting_for_touch_st: //name of state
        if(display_isTouched()){        //check if the screen is touched
            display_clearOldTouchData();// clear old touch data
            currentState = ad_timer_running_st;// go to the next state
        }
      break;//exit state
    case ad_timer_running_st: //name of state
        if (display_isTouched() && adcCounter == ADC_COUNTER_MAX_VALUE){ //check if the screen is touched
            currentState = auto_timer_running_st;// go to the next state
        }
        else if (!display_isTouched() && adcCounter == ADC_COUNTER_MAX_VALUE){
            clockDisplay_performIncDec();
            currentState = waiting_for_touch_st;// go to the next state
        }
      break;//exit state
    case auto_timer_running_st: //name of state
        if (display_isTouched() && autoCounter == AUTO_COUNTER_MAX_VALUE){//check if the screen is touched
            currentState = rate_timer_running_st;// go to the next state
        }
        else if (!display_isTouched()){
            clockDisplay_performIncDec();
            currentState = waiting_for_touch_st;// go to the next state
        }
      break;//exit state
    case rate_timer_running_st: //name of state
        if (display_isTouched() && rateCounter == RATE_COUNTER_MAX_VALUE){//check if
```

```
the screen is touched
            currentState = rate_timer_expired_st;// go to the next state
        }
        else if (!display_isTouched()){
            currentState = waiting_for_touch_st;// go to the next state
        }
      break;//exit state
    case rate_timer_expired_st: //name of state
        if (display_isTouched()){   //check if the screen is touched
            clockDisplay_performIncDec();
            currentState = rate_timer_running_st;// go to the next state
        }
        else if (!display_isTouched()){
            currentState = waiting_for_touch_st;// go to the next state
        }
      break;//exit state
    case add_second_to_clock_st: //empty
      break;//exit state
    default:
      printf("clockControl_tick state update: hit default\n\r");
      break;//exit state
  }
}
```