```c
/*************** STOPWATCH EXAMPLE ******************/
/***************  11 June 2019  *********************/
/****************************************************/

#include <msp430.h>

#include "peripherals.h"
#include "lecture.h"
#include "utils/test_runner.h"
#include "utils/ustdlib.h"

// Function Prototypes
void swDelay(char numLoops);
void startTimerA2(void);
void stopTimerA2(void);
void displayTime(unsigned long t);


// Global time count, storing number of timer ticks at 0.01s/tick
// NOTE:  Need to declare this as "volatile" since timer is modified in ISR
volatile unsigned long timer = 0;

#pragma vector=TIMER2_A0_VECTOR
__interrupt void TimerA2_ISR(void)
{
    timer++;

    // Non-stopwatch example:  blink LED at 1Hz (toggle twice per second)
    if ((timer % 50) == 0) {
        P1OUT ^= BIT0;
    }
}

// Main
void main(void)
{
    char timer_on = 0;
    unsigned long start_time = 0;

    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer.

    // *** System initialization ***
    configDisplay();
    initLaunchpadButtons();

    Graphics_clearDisplay(&g_sContext); // Clear the display
    Graphics_flushBuffer(&g_sContext);

    // Configure LED P1.0
    P1SEL &= ~BIT0;
    P1DIR |= BIT0;

    startTimerA2();
    _enable_interrupts(); // Enables global interrupts

    // Clear the display at startup
    Graphics_clearDisplay(&g_sContext);
    displayTime(0);
    Graphics_flushBuffer(&g_sContext);

    while (1)
    {
        char button_state = readLaunchpadButtons();

        if (button_state & BUTTON_S1) {
            timer_on = 1;
            start_time = timer; // Record timer value at start
        }

        if (button_state & BUTTON_S2) {
            timer_on = 0;
        }

        if (timer_on) {

                // Can compute time elapsed since start button was pressed
                // using start_time and current timer value
                // Note:  Important to save timer value (current_time) before
                // using it, as the value can change unexpectedly
                unsigned long current_time = timer;
                unsigned long elapsed_time = current_time - start_time;

                // Update display every 10 ticks
                // This is a way to prevent the display from
                // re-drawing more frequently than necessary
                if ((timer % 10) == 0) {
                    displayTime(elapsed_time);
                    Graphics_flushBuffer(&g_sContext);
                }
            }

        }

    }
}
```

```c
void displayTime(unsigned long curr_time)
{
    // First, get each portion of the display using integer math
    int total_sec =  curr_time / 100; // Seconds (integer part)

    int total_dec = curr_time % 100; // Hundredths (fractional part)

    int min = total_sec / 60; // Minutes
    int sec = total_sec % 60; // Seconds

    // Now we assemble these digits into a string
    // Here, we use usnprintf, a lightweight version of sprintf designed
    // for this system.
    // To use it, we create a buffer for the string and then define
    // format specifiers for each digit.  Here, "%02d" means we want
    // to print a two-digit integer with zero-padding (so 5min prints as 05)
    unsigned char str[9]; // mm:ss.hh = 8 chars + null terminator
    usnprintf(str, 9, "%02d:%02d.%02d", min, sec, total_dec);

    // NOTE 2:  If we clear the display every time, we may notice the display
    // flashing unnecessarily.  One way to resolve this is to *NOT* clear
    // the whole display and pass the parameter OPAQUE_TEXT to the function.
    // Unlike TRANSPARENT_TEXT (default), OPAQUE_TEXT clears the pixels under
    // the new string before drawing it, which is useful because we always
    // draw a fixed-size string to the same part of the display.
    //Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext, str, 9, 48, 15, OPAQUE_TEXT);
}

void startTimerA2(void)
{
    TA2CTL = TASSEL_1 | ID_0 | MC_1; // ACLK, Divide by 1, Up mode
    TA2CCR0 = 327;                   // 327 + 1 ACLK ticks = ~0.01s
    TA2CCTL0 = CCIE;                 // Enable capture/compare interrupt
}

void stopTimerA2(void)
{
    // Stop the timer
    TA2CTL = MC_0;     // Stop mode
    TA2CCTL0 &= ~CCIE; // Disable interrupts
}


// . . . Other demo functions omitted . . .

```