# Lab 3 Stopwatch (Mixed C/Assembly)
# Summer 2021

Lab location:   Online only

Lab Times

- T 8:00am – 10:30am, 7/6

Demo Due: The beginning of Lab 5 (7/13)

Report Due: 7/13, 11:59pm

## Introduction

The goal of this lab is for you to develop another simple, real-world embedded system application, and this time using **mixed C/assembly programming**. The learning objectives are:

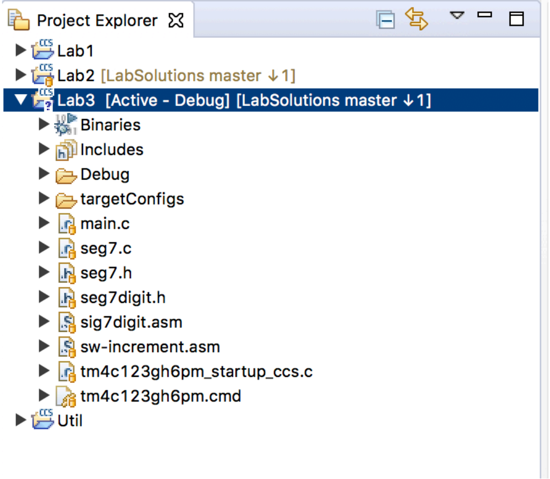
- Develop a stopwatch application
- **Exercise mixed assembly/C programming**
- Be familiar with assembly programming and debugging features of TI CCS
- Learn more about debugging in CCS on C code and assembly code

## Part 1:

**GitHub Repository Setup**: This is to be done by one group member.

1. In your working directory for ECE 266 (the folder used by GitHub Desktop), create a new folder named "Lab3". Download "lab3_main.c", "seg7digit.asm", "seg7digit.h", and "sw_increment.asm" from Lab 3 attachment to this folder. Then, copy "seg7.c" and "seg7.h" from your Lab 2 folder (or from Lab 2 attachment) to this folder.
2. Create CCS Project "Lab3" with external reference to the "Lab3" folder, configure the *include paths* and *library file paths*, then build the project.
3. Switch to GitHub Desktop, review the new changes. You should only see source code files. Note that CCS may add two new files, "tm4c123gh6pm.cmd" and "tm4c123gh6pm_startup_ccs.c".
4. Commit the new files into your local repository, and then push the changes into your GitHub repository. Log into your GitHub account to make sure the changes appear there.

This is the about the same procedure as you did in CCS Project Lab2 in the Lab 2 assignment. If necessary, review Lab 2 manual again for the detail. The CCS Project window will look like the following:



```
Project Explorer
▶ Lab1
▶ Lab2 [LabSolutions master ↓1]
▼ Lab3  [Active - Debug] [LabSolutions master ↓1]
    ▶ Binaries
    ▶ Includes
    ▶ Debug
    ▶ targetConfigs
    ▶ main.c
    ▶ seg7.c
    ▶ seg7.h
    ▶ seg7digit.h
    ▶ sig7digit.asm
    ▶ sw-increment.asm
    ▶ tm4c123gh6pm_startup_ccs.c
    ▶ tm4c123gh6pm.cmd
▶ Util
```

The other member(s): Log into your GitHub account and review the contents of the "Lab3" folder. Switch to GitHub Desktop and pull the changes. Switch to CCS, create CCS Project "Lab3" with external reference to the "Lab3" folder in your working directory (the same folder used by GitHub Desktop), configure the include paths and library file paths, and then build the project. At the end, your CCS Project window should look like the figure above.

**Identify and Approach the Programming Tasks.** The following procedure is *suggested*:

1. All members work together to identify the programming tasks. For example, there can be two major programming tasks (not exclusively) in this lab:
   a. Complete the stopWatchIncrement() function in sw-increment.asm (see below)
   b. Add a callback function in main.c to check pushbuttons

2. **All members discuss how to approach those tasks**. For example, what are the major execution steps in stopWatchIncrement()? Where is the code template for the callback function to check pushbuttons?

3. **Divide the tasks among members**. For example, one member may work on the assembly code and the other member may work on "main.c".

4. Each member completes the his/her tasks, tests the code if possible, and commits and pushes the code changes into the GitHub repository.

5. **Review the code changes from your partner(s)**. This step is important. Ask your partner(s) to explain the code whenever you have doubt.

   Note: In GitHub Desktop or github.com, you can review the changes between two versions of the codes.

6. **Test, debug, and revise codes (if any bug appears)**. Repeat until all bugs are fixed.

## Part 2: Stopwatch in Assembly

Attach the LaunchPad with the Grove base boosterpack and attach the Grove 4-digit 7-segment display to the **J10** jumper of the Gove base boosterpack.

Study the provide source code files, with a focus on the template assembly code in “sw_increment.asm”. You will revise this the assembly code to implement a running stopwatch.

You will need to use the following assembly instructions:  LDRB, STRB, ADD, CMP, BNE, and MOV. All those instructions have been used in the template assembly code.

[Optional] You may check the *ARMv7-M Architecture Reference Manual* (available on Blackboard under “Resources”), Section A7.7 A7 Instruction Details, for a detailed explanation of each of those instructions. The BNE instruction is B instruction with conditional code NE, which we will study in a later lecture.

**Revise the *stopWatchIncrement* function in “sw-increment.asm”.** The template code only updates the first (rightmost) digit of the 7-segment display. Revise it so that it updates all the four digits. The system should run as a stopwatch with a resolution of 0.01 second and a range of 0-60 seconds. The display should reset to “00:00” after “59:99”.

**Play with the debugging features of CCS when you debug this program**. For example, you may do the following:

- Step In or Step Over: Run a single assembly instruction.
- Set break point
- Review the contents of CPU registers
- Review the contents of memory

Notes:

- The TI ARM *assembly instructions* are almost the same as the ARM assembly in the textbook (Keil based).
- The TI ARM *assembly directives* are **very different** from the standard ARM assembly. Read the *ARM Assembly Language Tools v15.12.0.LTS User’s Guide*, Ch. 5, for detail. The guide is posted under “Resources” on Blackboard.

Demo your program to your TA. The demo should last for at least 60 seconds to test the wrap-around behavior.

## Part 3: Push Button Functions in C

The code in Part 2 does not yet make the system run as a full stopwatch. In this part, your system should run as in Part 2 but with the additional features as follows:

1. Initially, the system should be paused and show “00:00”.
2. The SW2 button works as a Start/Pause/Resume button. The user may push SW2

to start the stopwatch, push SW2 again to pause the stopwatch, and push SW2 again to resume the stopwatch.

3. The SW1 button works as a Reset button. When and only when the stopwatch is paused, the user may push SW1 to reset the stopwatch to "00:00" and then push SW2 to restart the stopwatch. If the stopwatch is running, pushing SW1 should have no effect.

You need to revise function `checkPushButton()` in "main.c". Typically, one may use a C switch-statement or a sequence of if-statements to implement an FSM (finite state machine) to track the state of the stopwatch.

*Hint*: This function may have to access two global variables, `sysState` and `seg7Display`. Careful use of global variables is acceptable and often necessary in embedded systems programming.

Demo your full stopwatch to your TA. The demo should last for at least 60 seconds to test the wrap-around behavior. Show that both pushbuttons work as required.

If you can demo for Part 3, you don't have to demo separately for Part 2.

## Demo Requirement

For this lab, at the time of demo, first show your source code to your TA and then explain how it works. Each of you may have to answer questions from your TA, which test your understanding of the working details. Your answer may affect your individual lab grade.

## Lab Submission

Your group should write a lab report that includes the following:

1. Describe the responsibility of each person in your group, and estimate the breakdown of contribution in percentage (e.g. 50% and 50% for a two-person group, or 33.3%, 33.3% and 33.3% for a three-person group).
   Note: If one contributes 40% or higher in a two-person group or 25% or higher in a three-person group, he or she may receive the same grade as the group gets.
2. Describe how you have revised the stopWatchIncrement() function in "sw-increment.asm".
3. Describe how you have revised the checkPushButton() function in "main.c".
4. How much flash memory does your program take? How much SRAM memory?
5. **Estimate how many hours in total you and your partners have worked on this lab.**

Make sure that you have pushed the latest version of your code into GitHub. **This step is required for you to get the grade.** Then, submit your 1) revised **sw_increment.asm**, 2) revised **main.c**, and 3) **lab report** on Blackboard under Lab 3 assignment.