1) From mycourses download this pdf, dxp_Lab7_a1.asm, and CapTouchOverview.pdf.
2) The **objective** of this lab is to learn how to read the state of the capacitive touch sensors.

3) **Part 1: Build, run and understand the dxp_Lab7_a1.asm code.**

4) Read the document CapTouchOverview to become familiar with how the capacitive "keypad" works. Use also your lecture notes. Refer as necessary to the MSP430 User's Guide and data sheet.

5) Remove J5 jumpers and install the capacitive booster pack. Store the jumpers by installing them on ONE header pin, with the other side "hanging off". This way you will not lose them.

6) Create a new assembly only project in your workspace named fmlxxxx_Lab7_a1. Create a copy of the assembly code dxp_Lab7_a1.asm in the project folder, and rename it fmlxxxx_Lab7_a1.asm. Build the project, enter debug mode, and run the program.

7) Set a breakpoint after the baseline value has been measured. Record it. Set another breakpoint on CheckAgain lable line. Compare meas_crt with meas_base.

8) The uC runs at a frequency of about 1.1 MHz. This is about 1 us per clock cycle. Count the number of cycles in the SW delay loop. Multiply by 1us. This is the total accumulation time. Divide this by the number of counts, both baseline and current, to find out the pin oscillation in both cases. From the graphs in the UG or DS find out the capacitances in both cases. Record all these calculations and measurements.

9) Change the SWdelay value to 6 and repeat step9.

10) Note: The code that is provided is basically using SWtimer as a fixed delay, to count the number of positive edges of the input frequency going into the timer. Every positive edge created by the touch sensor will increment the TAR register. Then after the fixed delay, we manually generate a "capture" event, so that the TAR register is captured into the TACCR1 register. This will now contain a number that represents the cap touch sensor frequency.

    In the provided code, there is an endless loop at the end, if the baseline is not equal to the captured value, it will think the button is pressed. This not always works as intended, because there is noise in the input signal. During the transition from "no press" to "press", your signal will drop below the baseline value several times. If you toggle every time you drop below the baseline, the system will assume the button is being pressed a second time when it is really not. Part 1, as it is given to you, will not actually "work" properly without modifying it. It is just intended for recording the baseline vs. a "button touched" measurement. To get it working as

intended, you will have to come up with a "threshold value" that is somewhere between the captured baseline and a "button pressed" value, and see if the current measurement is above or below that value. This will give you the best noise margins. You will then have to de-bounce the current measurement. There are many ways to do this; here are a couple of them:

A) Keep a counter that counts how many consecutive samples are under the threshold value. If the counter exceeds a particular value, then the button is being pressed. If the current sample is above the threshold, the counter is reset.

B) Use 2 threshold values, and keep track of the current button status. If the button is currently not being pressed, you are comparing the current samples with a "lower" threshold value. If it is lower than it, change the button status to "pressed". If the current button status is "pressed", then you compare current samples with a "higher" threshold value. If it is higher than it, then change the button status to "not pressed". This effectively gives you a "noise margin". Any samples that are between the 2 threshold values are basically ignored - these would be samples that come in when you are halfway through pressing the button.

11) **Now, modify the fmlxxxx_Lab7_a1.asm code to work as intended. The LED should TOGGLE each time the button is pressed. You will need a SW de-bouncing mechanism.** In your report, mention which threshold value you chose, and which de-bouncing mechanism you used.

12) **Part 2: Modify the fmlxxxx_Lab7_a1.asm code to fmlxxxx_Lab7_a2.asm and change the functionality to:** keeping the LED on or off for as long as the "key" is "pressed" or "depressed". This functionality will not require a de-bouncing mechanism.

13) **Part 3: Modify the fmlxxxx_Lab7_a1.asm code to fmlxxxx_Lab7_c1.c.** Create the same functionality in C code. Use previous C code examples as guides and templates. A while(1) loop and a subroutine call will be the main structures of your code. The LED should TOGGLE each time the button is pressed. You will need de-bouncing.

14) **Part 4: Modify the fmlxxxx_Lab7_a2.asm code to fmlxxxx_Lab7_c2.c.** Create the same functionality in C code. The LED should stay on for as long as the "key" is "pressed".

15) Make sure you write the report and upload it along with your project archives on mycourses. As time permits, demo the intermediate steps to the TA.

16) **Grading**:

a. Part 1 = 10 points

b. Part 2 = 10 points

c. Part 3 = 10 points

d. Part 4 = 10 points.