ECSE426 - Microprocessor Systems

Fall 2013

Lab 4: Multithreaded, interrupt-driven multi-sensor detection

(1) The Lab

This lab is aimed at designing a multithreaded system that uses CMSIS-RTOS for initiating and maintaining multiple threads of computation and the exception types executed concurrently. You will be using the temperature and accelerometer sensors to perform a seamless superset of sensing and displaying functions from previous labs. The LED lights will interchangeably display the processed outputs of the thermometer and the accelerometers, where you will switch between the two *modes* by taping the board. Each mode will have two sub-modes which will be triggered by pressing the user button. You have to design a system where all the functions are seamlessly integrated in multiple threads of computation and Interrupt Service Routines (ISRs)..

The system will have two main modes, which switch in between when a **tap** is detected:

1. **Temperature mode:** In this mode, LEDs will either display the change trend in temperature over a scale of 8 degrees with wrap around, or flash all LEDs (as in Lab 2). A button press will change from one sub-mode to another. The measurement of the temperature will be done concurrently in the background with determining the orientation of the board in the other mode. This means that the computation will be running all the time in the background, but will only be displayed when the temperature mode is selected.

2. **Accelerometer mode:** In this mode, the default function conveys information about the board's tilt angle. Similar to lab three, you will have to convey this change by controlling the speed of the LED flashing. You have to divide the available angle range into regions and each region will have a different discreet flashing speed. When the button is pressed, you will switch to the hardware PWM mode.

The threads of computation should be designed using CMSIS-RTOS primitives using threads, timers, ISRs and any required OS services (mutexes, semaphores and queues ... etc). In filtering the sensor data, you will use the moving average filter as before. The sensor data should also be calibrated. An important property is that the system should perform regardless of combination of taps, clicks, the current orientation of the board and the temperature change, without any assumptions on current orientation (should work if placed upside-down), temperature change, or any of the external changes.

You should ensure that a button click does not adversely impact the tap detection, i.e., that a click on the user button does not get interpreted as a tap. As in the last lab, you should do the minimal amount of work in ISRs for best results.

(2) Requirements checklist

To summarize, the tasks you need to do are:

1. Calibration and filtering for the sensors (retain the sampling rates from previous labs),
2. Detect **single taps** and use them to switch between the two main modes (Temp and Accl),
3. Temperature mode, upon button press switch between sub-modes:
    - Temperature trend display (8 degrees with wrap around),
    - Flashing mode,
4. Accelerometer mode, upon button press switch between sub-modes
    - Control LED flashing speed based on tilt
    - Hardware PWM
5. All above functions are to be implemented using CMSIS-RTOS.
6. You have to present a graph which shows how your application is divided into threads, functions, ISRs … etc.

(3) Debugging in real time

Since you will be using CMSIS-RTOS and the underlying OS (RTX), you have to be capable of debugging your code and understanding how to simplify it in face of a sizeable body of OS code. Furthermore, most processing power will be directed towards periodic tasks - interrupt- and exception-driven. When validating and debugging your solution, it is especially important to use non-obtrusive methods provided by Cortex M4 and Keil feature set. Your real-time tracing should not adversely affect the program execution.

Refer to the tutorial slides to learn about Keil support for visualizing and debugging thread operation. Keil offers a visual display which will show how threads are temporally scheduled. It also has tools to show the states of each thread, its total stack usage among other important parameters.

(4) Proving and demonstrating your solution

You are expected to demonstrate the resulting program in real time, as per above specification, as well as the proper interleaving of independent processing threads and asynchronous events. In addition to using the SWD debug interface, you should be ready to provide the evidence that your solution meets the specification. **Your solution should also be safe that is, when dealing with multiple threads of operation**. For example, you should satisfy the mutual exclusion access property on shared variables if necessary. It should also meet the same timing constraints as before (sampling frequencies of Lab 2 and Lab 3).

You are required to keep the continuity of operation and to determine angles from the data available, even when the tap is detected.

(5) Demonstration

You will need to demonstrate that your program is correct, and explain in detail how you guarantee the specified operation, and how you tested your solution. You will be expected to demonstrate a functional program and its operation performs in all situations. The final demonstration will be on **Monday**, November, 11th. **Note that you have 8 days to carry out this task**. As after that, you will be given three weeks to carry out the project. This should be an ample time to finish as you are not writing any major code or using any peripherals. The lab is about re-organizing your code to make use of RTOS.

(6) Lab Report

The report must include structured explanation of the new work done in the lab. The report should contain the explanation and validation of all major design decisions (e.g., task division into threads, filtering, sampling, detecting, displaying and handling tapping) throughout the lab.

The report is to be submitted by Wednesday, November. 13th.