Exercises 2IN60.4

Today we will continue experimenting with the μC-OS-II real-time operating system running on the Freescale EVB9S12XF512E board.

4.1 Limiting the time that interrupts are disabled

In this exercise you will gain insight into the consequences of non-preemptive task execution introduced by critical sections, and learn how to mitigate them. The CodeWarrior project for this exercise is in the directory `exercise4_1`.

1. In this exercise we continue with the program from the last exercises. The `main.c` and `ATD_driver.c` files are taken from the project in the `exercise3_2_solution` directory[1]. Compare these files with your own results from the last exercises. You are welcome to use your own `main.c` and `ATD_driver.c` implementations in this exercise, if you are convinced that they are working correctly.

2. To avoid interference between the two tasks sharing the ATD converter, we have made the ATD conversion atomic by disabling interrupts inside the function `ATDReadChannel()`. **Using the simulator, measure the length of the critical section.**

3. μC-OS-II keeps track of time by means of a tick counter which is incremented every time the timer interrupt is handled. Disabling interrupts for a long time may lead to missed interrupts, and consequently wrong tick counter value (i.e. the tick counter not representing the actual time). Therefore, we would like to check whether interrupts are missed or not.

   We can do it by comparing the tick count (returned by `OSTimeGet()`) before and after the conversion to see how many ticks were recorded during the conversion. Since both tasks use the ATD converter, we have a choice where to measure the ticks. **In which task would you count the ticks? By how many ticks do you expect the before and after counts to differ? Motivate your answer.**

   Hint: use the information you have obtained in Step 2, and consult Chapter 16 in [Labrosse, 2002] (available on the course website) for the specification of `OSTimeGet()`.

4. The task that you have selected in step 3 should now be extended as follows: if the ATD conversion took fewer ticks than expected (meaning that ticks were missed) then led `D24` should be turned on, otherwise it should be turned off. Write on a piece of paper the new definition of the task.

5. **Before you implement and run the program, write down how you expect the leds to behave. Motivate your answer.**

6. Implement the tasks specified in Step 4 and verify your prediction by running the program on the board.

[1] The bit resolution of the ATD conversion has changed from 8 bits to 12 bits.

7. **Describe the problem, propose one solution.**

   Hint: in the current implementation of `ATDReadChannel()` atomicity was achieved by disabling the interrupts. Atomicity with respect to other tasks can be achieved differently.

8. **Implement your proposed solution from the previous step.**

9. Verify your implementation by running the program on the board.

10. In step 7 you have identified a problem with disabling interrupts during sensor reading and proposed a solution. **What drawbacks does your solution have? Describe a situation illustrating the problem.**