2IN70 - Track 2: Real-Time Architectures
Practical Training - Exercises 5 -

Today we will experiment with the task synchronization primitives in the μC/OS-II real-time operating system running on the Freescale EVB9S12XF512E board.

5.1 Limiting priority inversion

In this exercise you will gain insight into limiting priority inversion when only a subset of tasks shares a mutually-exclusive resource. The CodeWarrior project for this exercise is in the directory `exercise5_1`. *Note that the RTI interrupt frequency is changed to 2 kHz, meaning that the tick period is 0.5ms. An ATD conversion takes about 2.15ms. Keep these in mind when you analyze the timing of the tasks.*

1. Look at the task specification and definition in `main.c` and *before running it on the board* write down on a piece of paper at what times you expect `LED_D26` to toggle in the time interval [0s, 100s].

   Note: you can also provide a formula, or a set of formulas, describing when you expect the led to toggle.

2. **Draw a timeline of the task execution in the time interval [0ms, 1005ms].** Do not forget to include the relevant timing information, e.g. the time unit on the axis, the relevant time intervals, etc.

3. Run your program on the board and verify your prediction.

4. In Exercise 4.1 we made sure that the ATD conversion inside `ATDReadChannel()` is interruptible by the timer interrupt, but not preemptable by *any* task. **Is this requirement for non-preemption necessary? If not, how would you relax the requirement?**

   Hint: Look at the task definition in `main.c` and the resources they use.

5. **How would you implement your new requirement from Step 4? Describe briefly what would need to change in the current implementation.**

   Hint: use one of the μC/OS-II task synchronization primitives. You may like to consult Chapter 16 in [Labrosse, 2002] (available as documentation) for the specification of the primitives. Describe which functions you would call, where you would call them, and with what parameters. Think about task priorities in `main.c` when choosing the parameters for creating the primitive.

6. **Implement your proposed design.**

7. *Before running it on the platform*, describe on a piece of paper how you expect the leds to behave.

8. Run your program on the board and verify your prediction.

9. **Draw a timeline of the task execution in the time interval [0ms, 1005ms].**

10. **Is it possible for your implementation to deadlock? Motivate your answer.**