IL2206 EMBEDDED SYSTEMS

Laboratory 1B : Real-Time Scheduling

Version 1.0.0 (September 10, 2025)

1 Objectives

In this laboratory students shall deepen their understanding of real-time scheduling.

The laboratory tasks can be conducted with different execution platforms by either using

- using the Real-Time Annex of Ada, which requires that the student has access to an own computer, which can run in supervisor mode; or
- using FreeRTOS on the Embedded Systems Lab-Kit board from KTH.

2 Preparation

It is very important that students are well-prepared for the labs, since both lab rooms and assistants are expensive and limited resources, which shall be used efficiently. The laboratory will be conducted by groups of two students. However, each student has to understand and explain the developed source code and the laboratory tasks to pass the laboratory.


Documentation
All program code shall be well-structured and well-documented. The language used for documentation is English.

2.1 Installation of Ada or FreeRTOS

Please check the installation instructions on the Canvas page. Students, who use Ada will run all programs on their own computer, while students who use FreeRTOS will execute the programs on the Embedded Systems Lab-Kit development board.

2.2 Code Skeletons

KTH provides code skeletons for the laboratory, which can be downloaded from the course page in Canvas.

2.3 Executing Ada Programs using the Real-Time Annex

Since this laboratory uses the real-time annex, and requires to only use one core, special care has to be taken if you conduct this laboratory using Ada. Please follow the guidelines stated in the following box.

Executing Ada Programs

- Ada programs will in general use all cores. If programs shall run on a single core, it has to be enforced by the user. In Linux the user can use the command `sudo taskset -c 0 ./program_name` to enforce execution of a single core.
- If you use the real-time annex in Ada, the programs need to be run in supervisor mode for the correct timing!

NOTE! Programs using the real-time annex will not run correctly on KTH computers, since students do not have root access.

2.4 Executing FreeRTOS Programs on the Embedded Systems Lab-Kit

Student should create a FreeRTOS project following the guidelines for the Embedded Systems Lab-Kit[1].

Setting the number of cores in `FreeRTOSConfig.h`

The RP2350 on the development board has two cores. Depending on the settings in the `FreeRTOSConfig.h` file, your program will use one or two cores. Please check, that you have set the right number of cores in the `FreeRTOSConfig.h` file by modifying the definition of `configNUMBER_OF_CORES`.

3 Laboratory Tasks

NOTE! The instructions are written for Ada. However, all tasks can be conducted using FreeRTOS and the corresponding code skeletons are available on the Canvas page for this laboratory.

[1] `https://gits-15.sys.kth.se/mabecker/ES-Lab-Kit`

3.1 Periodic Tasks in Ada

A real-time Ada program, where six tasks with fixed priorities are scheduled priority-driven, produces the following output on a single core computer. Here, WCRT denotes the measured Worst Case Response Time.

```
> sudo taskset -c 0 ./periodictasks_analyse_priority
Warm-Up - No task released for  100 Milliseconds
Task 4- Release:  0.200, Completion: 0.374, Response: 0.174, WCRT: 0.174, Next Release:  0.900
Task 5- Release:  0.400, Completion: 0.582, Response: 0.182, WCRT: 0.182, Next Release:  1.000
Task 6- Release:  0.600, Completion: 0.702, Response: 0.102, WCRT: 0.102, Next Release:  1.000
Task 1- Release:  0.800, Completion: 0.946, Response: 0.146, WCRT: 0.146, Next Release:  1.700
Task 2- Release:  1.000, Completion: 1.089, Response: 0.089, WCRT: 0.089, Next Release:  1.500
Task 6- Release:  1.000, Completion: 1.179, Response: 0.179, WCRT: 0.179, Next Release:  1.400
Task 3- Release:  1.200, Completion: 1.289, Response: 0.089, WCRT: 0.089, Next Release:  2.000
Task 5- Release:  1.000, Completion: 1.359, Response: 0.358, WCRT: 0.358, Next Release:  1.600
Task 4- Release:  0.900, Completion: 1.396, Response: 0.496, WCRT: 0.496, Next Release:  1.600
Task 2- Release:  1.500, Completion: 1.589, Response: 0.089, WCRT: 0.089, Next Release:  2.000
Task 6- Release:  1.400, Completion: 1.592, Response: 0.192, WCRT: 0.192, Next Release:  1.800
Task 1- Release:  1.700, Completion: 1.788, Response: 0.088, WCRT: 0.146, Next Release:  2.600
Task 6- Release:  1.800, Completion: 1.889, Response: 0.089, WCRT: 0.192, Next Release:  2.200
Task 5- Release:  1.600, Completion: 1.897, Response: 0.297, WCRT: 0.358, Next Release:  2.200
Task 4- Release:  1.600, Completion: 1.986, Response: 0.386, WCRT: 0.496, Next Release:  2.300
Task 3- Release:  2.000, Completion: 2.096, Response: 0.096, WCRT: 0.096, Next Release:  2.800
Task 2- Release:  2.000, Completion: 2.185, Response: 0.185, WCRT: 0.185, Next Release:  2.500
Task 6- Release:  2.200, Completion: 2.322, Response: 0.122, WCRT: 0.192, Next Release:  2.600
Task 5- Release:  2.200, Completion: 2.413, Response: 0.213, WCRT: 0.358, Next Release:  2.800
Task 2- Release:  2.500, Completion: 2.588, Response: 0.088, WCRT: 0.185, Next Release:  3.000
Task 4- Release:  2.300, Completion: 2.591, Response: 0.291, WCRT: 0.496, Next Release:  3.000
Task 1- Release:  2.600, Completion: 2.729, Response: 0.129, WCRT: 0.146, Next Release:  3.500
Task 3- Release:  2.800, Completion: 2.889, Response: 0.089, WCRT: 0.096, Next Release:  3.600
Task 6- Release:  2.600, Completion: 2.908, Response: 0.308, WCRT: 0.308, Next Release:  3.000
Task 5- Release:  2.800, Completion: 2.998, Response: 0.198, WCRT: 0.358, Next Release:  3.400
Task 2- Release:  3.000, Completion: 3.090, Response: 0.090, WCRT: 0.185, Next Release:  3.500
Task 6- Release:  3.000, Completion: 3.180, Response: 0.180, WCRT: 0.308, Next Release:  3.400
Task 4- Release:  3.000, Completion: 3.270, Response: 0.270, WCRT: 0.496, Next Release:  3.700
Task 2- Release:  3.500, Completion: 3.589, Response: 0.089, WCRT: 0.185, Next Release:  4.000
Task 3- Release:  3.600, Completion: 3.689, Response: 0.089, WCRT: 0.096, Next Release:  4.400
Task 1- Release:  3.500, Completion: 3.768, Response: 0.268, WCRT: 0.268, Next Release:  4.400
Task 6- Release:  3.400, Completion: 3.795, Response: 0.395, WCRT: 0.395, Next Release:  3.800
Task 6- Release:  3.800, Completion: 3.889, Response: 0.089, WCRT: 0.395, Next Release:  4.200
Task 5- Release:  3.400, Completion: 3.975, Response: 0.575, WCRT: 0.575, Next Release:  4.000
Task 2- Release:  4.000, Completion: 4.089, Response: 0.089, WCRT: 0.185, Next Release:  4.500
Task 5- Release:  4.000, Completion: 4.179, Response: 0.179, WCRT: 0.575, Next Release:  4.600
Task 6- Release:  4.200, Completion: 4.289, Response: 0.089, WCRT: 0.395, Next Release:  4.600
Task 4- Release:  3.700, Completion: 4.334, Response: 0.634, WCRT: 0.634, Next Release:  4.400
Task 3- Release:  4.400, Completion: 4.547, Response: 0.147, WCRT: 0.147, Next Release:  5.200
Task 2- Release:  4.500, Completion: 4.636, Response: 0.136, WCRT: 0.185, Next Release:  5.000
Task 1- Release:  4.400, Completion: 4.726, Response: 0.326, WCRT: 0.326, Next Release:  5.300
Task 6- Release:  4.600, Completion: 4.817, Response: 0.217, WCRT: 0.395, Next Release:  5.000
Task 5- Release:  4.600, Completion: 4.907, Response: 0.307, WCRT: 0.575, Next Release:  5.200
Task 4- Release:  4.400, Completion: 4.997, Response: 0.597, WCRT: 0.634, Next Release:  5.100
Task 2- Release:  5.000, Completion: 5.090, Response: 0.089, WCRT: 0.185, Next Release:  5.500
Task 6- Release:  5.000, Completion: 5.179, Response: 0.179, WCRT: 0.395, Next Release:  5.400
Task 3- Release:  5.200, Completion: 5.289, Response: 0.089, WCRT: 0.147, Next Release:  6.000
Task 1- Release:  5.300, Completion: 5.390, Response: 0.090, WCRT: 0.326, Next Release:  6.200
Task 6- Release:  5.400, Completion: 5.489, Response: 0.089, WCRT: 0.395, Next Release:  5.800
Task 2- Release:  5.500, Completion: 5.590, Response: 0.090, WCRT: 0.185, Next Release:  6.000
Task 5- Release:  5.200, Completion: 5.648, Response: 0.448, WCRT: 0.575, Next Release:  5.800
Task 4- Release:  5.100, Completion: 5.718, Response: 0.617, WCRT: 0.634, Next Release:  5.800
Task 3- Release:  6.000, Completion: 6.089, Response: 0.089, WCRT: 0.147, Next Release:  6.800
Task 2- Release:  6.000, Completion: 6.179, Response: 0.179, WCRT: 0.185, Next Release:  6.500
Task 6- Release:  5.800, Completion: 6.182, Response: 0.382, WCRT: 0.395, Next Release:  6.200
Task 1- Release:  6.200, Completion: 6.289, Response: 0.089, WCRT: 0.326, Next Release:  7.100
Task 6- Release:  6.200, Completion: 6.379, Response: 0.179, WCRT: 0.395, Next Release:  6.600
Task 5- Release:  5.800, Completion: 6.452, Response: 0.652, WCRT: 0.652, Next Release:  6.400
Task 2- Release:  6.500, Completion: 6.589, Response: 0.089, WCRT: 0.185, Next Release:  7.000
Task 6- Release:  6.600, Completion: 6.689, Response: 0.089, WCRT: 0.395, Next Release:  7.000
Task 5- Release:  6.400, Completion: 6.720, Response: 0.320, WCRT: 0.652, Next Release:  7.000
Task 3- Release:  6.800, Completion: 6.889, Response: 0.089, WCRT: 0.147, Next Release:  7.600
Task 4- Release:  5.800, Completion: 6.898, Response: 1.098, WCRT: 1.098, Next Release:  6.500
Task 4- Release:  6.500, Completion: 6.988, Response: 0.488, WCRT: 1.098, Next Release:  7.200
Task 2- Release:  7.000, Completion: 7.130, Response: 0.130, WCRT: 0.185, Next Release:  7.500
```

```
Task 1- Release:  7.100, Completion: 7.221, Response: 0.120, WCRT: 0.326, Next Release:  8.000
Task 6- Release:  7.000, Completion: 7.310, Response: 0.310, WCRT: 0.395, Next Release:  7.400
Task 6- Release:  7.400, Completion: 7.489, Response: 0.089, WCRT: 0.395, Next Release:  7.800
Task 5- Release:  7.000, Completion: 7.490, Response: 0.489, WCRT: 0.652, Next Release:  7.600
Task 2- Release:  7.500, Completion: 7.590, Response: 0.090, WCRT: 0.185, Next Release:  8.000
Task 3- Release:  7.600, Completion: 7.690, Response: 0.090, WCRT: 0.147, Next Release:  8.400
Task 5- Release:  7.600, Completion: 7.780, Response: 0.180, WCRT: 0.652, Next Release:  8.200
Task 6- Release:  7.800, Completion: 7.889, Response: 0.089, WCRT: 0.395, Next Release:  8.200
Task 4- Release:  7.200, Completion: 7.939, Response: 0.739, WCRT: 1.098, Next Release:  7.900
Task 2- Release:  8.000, Completion: 8.089, Response: 0.089, WCRT: 0.185, Next Release:  8.500
Task 1- Release:  8.000, Completion: 8.178, Response: 0.178, WCRT: 0.326, Next Release:  8.900
Task 6- Release:  8.200, Completion: 8.289, Response: 0.089, WCRT: 0.395, Next Release:  8.600
Task 5- Release:  8.200, Completion: 8.379, Response: 0.179, WCRT: 0.652, Next Release:  8.800
Task 4- Release:  7.900, Completion: 8.386, Response: 0.486, WCRT: 1.098, Next Release:  8.600
Task 3- Release:  8.400, Completion: 8.512, Response: 0.112, WCRT: 0.147, Next Release:  9.200
```

Try to order the tasks according to their task priority. Give a motivation how you have arrived at your conclusion.

◯ 3.1 completed

3.2 Rate-Monotonic Scheduling

Given is the following set of periodic tasks:

$$\Gamma_1 = \{\tau_1(100, 300, 100, 300), \tau_2(100, 400, 100, 400), \tau_3(100, 600, 100, 600)\}$$

where the times are given in milliseconds. A periodic task τ_i is defined as a tuple $\tau_i(\phi_i, T_i, C_i, D_i)$, where ϕ_i denotes the phase, T_i the period, C_i the computation time, and D_i the relative deadline.

1. Calculate the utilisation and draw the rate-monotonic schedule for this set of tasks for one hyperperiod.
2. Given is the skeleton program `periodictasks_priority.adb`[2]. Run the program for some time iterations on a *single core* [3] and calibrate the program by adjusting the constant `Calibrator`[4], so that the measured worst case execution time for the running task is close to the given computation time.
3. Implement the periodic task set Γ_1 in Ada, so that the tasks are scheduled rate-monotonically. Build your implementation on the calibrated skeleton program `periodictasks_priority.adb`.
   (a) Save the program as `rms.adb`. Execute the program and validate the expected behaviour.
   (b) Save the output from one simulation in electronic format and provide it as part of your solution.
   (c) Run the program several times for a few hyperperiods. Does the program follow the schedule from Task 1? Try to explain possible deviations between the schedule in reality and the theoretical one.
   (d) If possible, run the program `rms.adb` using all cores on your computer[5]. Explain the result of the execution.

[2] The corresponding FreeRTOS skeleton is available on the Canvas page for the course.

[3] See the instructions in Section 2.3 and Section 2.4.

[4] The FreeRTOS version of the laboratory does not require calibration, since it uses a timed waiting function that is provided by the board support package (BSP).

[5] In Linux the cores 0 to 3 will be used, if you use the command `sudo taskset -c 0,1,2,3 ./rms2`. For more information on your cores, run `sudo less /proc/cpuinfo`.

4. Add now an additional task $\tau_4 = (100, 1200, 200, 1200)$.

   (a) Draw the schedule for the program for one hyperperiod.

   (b) Save the program as `rms2.adb`. Run the program several times for a few hyperperiods. Compare the resulting schedule with the one of Task 4a. Explain possible deviations.

   NOTE: In case you see no deadline violations, increase the length of the computation times by adjusting the constant `Calibrator` or by increasing the execution time for task τ_4.

   (c) Save the output from one simulation in electronic format and provide it as part of your solution. Compare it with the theoretical results of subtask 4a and explain the resulting schedule.

◯ 3.2 completed

3.3 Overload Detection with Watchdog Timer

In order to be able to detect an overloaded system, add both a watchdog timer task and one additional helper task to your program `rms2.adb` and save it as `overloaddetection.adb`.

The watchdog timer and the helper task shall communicate with each other using a *suitable communication mechanism*[6]. An overload happens when there is not enough free CPU capacity to run all the user tasks until completion within a hyperperiod.

Study the principal functionality of a watchdog timer in the lecture notes and design the watchdog timer. Then think about how an additional helper task should be designed, so that the watchdog timer task can detect an overloaded system. In case of an overload, the watchdog timer shall output a warning. Make a sketch of your design and be prepared to explain the functionality of your solution to the course staff. To solve this task, it is important to understand how the fixed-priority scheduling algorithm determines which task should be scheduled and executed, and when these decisions are taken. Thus, you also have to think about which priorities should be given to the watchdog timer task and the helper task.

1. Run the system with overload detection and the task set $\mathbf{\Gamma_1} = \{\tau_1, \tau_2, \tau_3\}$ as described in Section 3.2-3.

2. Run the system with overload detection and the task set $\mathbf{\Gamma_2} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ as described in Section 3.2-4. Did you observe a system overload? When did it occur? If not, increase the workload. Explain the results.

◯ 3.3 completed

3.4 Mixed Scheduling

Add now three background tasks to the program from Task 3.2-3, which run on a low priority and are scheduled in a round-robin fashion. Each background task has an execution time of 100 milliseconds. Implement this system as `mixedscheduling.adb` using the high-priority task set $\mathbf{\Gamma_1} = \{\tau_1, \tau_2, \tau_3\}$.

[6] Examples for communication mechanisms are protected object, rendezvous, message queue, or semaphore.

In order to enable mixed scheduling in Ada, use the following pragmas for the high-priority and the background tasks[7].

```
pragma Priority_Specific_Dispatching(
               FIFO_Within_Priorities, 2, 30);
pragma Priority_Specific_Dispatching(
               Round_Robin_Within_Priorities, 1, 1);
```

Calculate the time, when the first background task should be executed in theory and compare with the practical result.

◯ 3.4 completed

3.5 Multi-Processor Execution

Increase the number of processors that Ada (or FreeRTOS) uses, for instance to 2. Run the programs `overloaddetection` of Task 3.3-2 and `mixedscheduling` of Task 3.4. How does this change affect the execution compared to a single-processor run? Analyse and explain the execution of the programs.

◯ 3.5 completed

4 Examination

Demonstrate the programs that you have developed for the laboratory staff during your laboratory session. Be prepared to explain your program in detail.

Each student in a student group shall be able to explain all parts of the laboratory and the written code. In order to pass the laboratory the student must have completed all tasks of Section 3 and have successfully demonstrated them for the laboratory staff.

[7] In FreeRTOS, if tasks have the same priority, the default behaviour is that they will be scheduled in a round-robin fashion.