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

Today we will start with programming the MC9S12XF512 micro-controller. In particular, we will experiment with different variants of the cyclic executive. We will execute these programs using the Freescale CodeWarrior simulator.

2.1 As Fast As Possible (AFAP)

In this exercise you will implement two tasks and execute them using the AFAP approach. Each task reads a sensor connected via the ATD converter and actuates a led in case the read sensor value crosses a threshold. The CodeWarrior project for this exercise is in the directory `exercise2_1`.

1. **The `main.c` file contains a specification of the two tasks: `Task1()` and `Task2()`. Implement these tasks, making use of the `ATDReadChannel()` function introduced during the lecture.**

   Hint: you may like to look at the `Led_driver.h` and `Led_driver.c` files inside the `Drivers` directory for functions to toggle and set the leds.

2. **Implement the `main()` function, making sure that `Task1()` is executing before `Task2()` according to the AFAP cyclic executive approach.**

Copy your modified code from the `main.c` file and paste into your report as answers to steps 1 and 2.

Let $T_i^{\min}$ and $T_i^{\max}$ be the minimum and maximum inter-arrival time between two consecutive jobs of task τ_i, respectively. Activation jitter for task τ_i is defined in terms of the maximum and minimum inter-arrival time between its two consecutive jobs.

3. **Measure the minimum and maximum execution times (in cycles) of `Task1` and `Task2`.**

   Hint: add `asm nop;` instructions around the task invocations inside the `main()` function and place breakpoints at these instructions to measure the execution times of the tasks. For example, to measure the execution time of `Task1()`, add `asm nop;` instructions around it:

   ```
   asm nop;
   Task1();
   asm nop;
   ```

   and place the breakpoints at the two `asm nop;` instructions. Remember: the execution stops *just before executing the instruction of the breakpoint*. When reporting the measurement, make sure to take into account the execution time of the `NOP` instruction (you can measure it by stepping over the single assembly instruction in the simulator or look it up in the HCS12 manual [2].

Note that the simulated ATD converter always reads a value 0 from all the ports. Think how you can measure all possible paths through your code and **explicitly describe in your report how you forced the tasks to follow the paths**.

4. **Give a formula for the inter-arrival time between two consecutive jobs for `Task1` and `Task2`.**

   Hint: you may like to consult an explanation of drift in [1].

5. **Derive the formula for the activation jitter for the k^{th} job of `Task1` and `Task2`.**

2.2 Time-driven AFAP

In this exercise you will make the control loop periodic. The CodeWarrior project for this exercise is in the directory `exercise2_2`.

1. Copy your task definitions and control loop from Exercise 2.1 into the `main.c` file in this project.

2. The Freescale HCS12 instruction set provides instructions `STOP` and `WAI` which can be used to suspend the processor. These instructions are described in Section 5.27 of the HCS12 manual [2]. **Use one of these instructions to implement Time-driven AFAP, i.e. to activate the task sequence periodically.** Check the implementation of `ATDReadChannel()` in the`ATD_driver.c` file for the syntax for writing assembly instructions in C code.

3. The Freescale MC9S12XF512 micro-controller can generate a Real-Time Interrupt at a fixed frequency, which is derived from the main CPU clock by means of a divider. The frequency of the timer is set in the `CPUInitRTI()` function in the `cpu.c` file. It is currently set to 1KHz. **Consult Section 2.3.2.8 in the MC9S12XF512 manual [3] and change the frequency to 2Hz.**

2.3 Activation jitter and drift

In this exercise you will investigate drift. The CodeWarrior project for this exercise is in the directory `exercise2_3`.

1. **The `main.c` file contains a specification of the two tasks: `Task1()` and `Task2()`. Implement these tasks.**

2. The `main()` function contains a time-driven AFAP control loop, which iterates over the task sequence 1000 times. Place brake points at the `asm nop;` instructions around the control loop and **measure the activation jitter of the 1001st job of `Task1()`**.

3. Modify the `main()` function to implement a simple AFAP control loop, which iterates over the task sequence 1000 times. Place brake points at the `asm nop;` instructions around the control loop and **measure the activation jitter of the 1001st job of `Task1()`**.

4. **Does the AFAP or the time-driven AFAP control loop suffer from drift? Motivate your answer.**

References

[1] R.J. Bril and M.J. Holenderski. Jitter and Drift. Internal note for course 2IN60 of TU/e, WIN, SAN, Technische Universiteit Eindhoven, November 2011.

[2] Freescale Semiconductor. S12CPUV2 Reference Manual. Document of Freescale Semiconductor, March 2006. Revision 4.0.

[3] Freescale Semiconductor. MC9S12XF512 Reference Manual. Document of Freescale Semiconductor, November 2010. Revision 1.20.